THE UNIVERSITY OF ALBERTA

DESIGN OF AN ON-LINE UDC LIBRARY AUTOMATION SYSTEM

by

© JOHN ALLEN BENBOW

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1974

THE UNVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled " DESIGN OF AN ON-LINE UDC LIBRARY AUTOMATION SYSTEM " submitted by John Allen Benbow in partial fulfilment of the requirements for the degree of Master of Science.

W S Hom
Supervisor

To-yat Cheung

Susan Bertram

H Fo Chan

DATED May 21, 1974

ABSTRACT

To date, many on-line library systems have been designed and implemented. However, very little work has been done to design and implement an economical and efficient on-line integrated UDC-based library system which uses to advantage the characteristics of the UDC notation.

This study is involved with the design considerations of an on-line UDC based library system. It represents the design, implementation, testing and evaluation of a file structure that is economical in terms of disk storage requirements and in terms of C.P.U. time needed to provide very short response times for on-line transactions. The proposed file design uses features of the UDC notation to provide an economical thesaurus for the integrated system.

Most of the effort in the investigation was expended in the design of a new method for the construction of the dictionary files for searchable words, and their corresponding inverted index files. An evaluation of these files in terms of storage requirements and file access times shows the power of the proposed design.

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Professor H.S. Heaps, my supervisor, for his advice, criticism, and guidance throughout the duration of this research; to Mrs. G.A. Cooke and her colleagues at the Boreal Institute for their helpful suggestions and guidance; and to Professor D.M. Heaps for her continuing interest and encouragement. In addition, I would like to express my gratitude to the following for their assistance: Dr. I.N. Chen, Dr. S. Bertram, Dr. T.Y. Cheung, and my fellow graduate students, especially E.J. Schuegraf.

Above all I would like to thank my wife, Ruth, to whom this thesis is dedicated, for her patience, understanding, assistance, and constant encouragement.

This research was partically supported by grants from the National Research Council of Canada

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
1.0	Introduction	1
1.1	Automation of a UDC Based Library	1
1.1.1	The Performance of an On-Line System	2
1.1.2	The Construction of a Thesaurus for a UDC-Based Library System	3
1.2	Statement of the Problem	3
1.3	The Boreal Institute for Northern Studies	4
1.3.1	The Library of the Boreal Institute for Northern Studies	5
1.4	An Overview of the Automated Library System	6
1.4.1	The Searching Subsystem	6
1.4.2	The Classifying Subsystem	10
1.4.3	The Acquisition Subsystem	12
1.4.4	The Real-Time Circulation Subsystem	12
II.	COMPRESSED DATA BASE	16
2.0	Introduction	16
2.1	The Selection of a Compression Coding Scheme	16
2.2	A Description of the Selected Coding Scheme	17
2.3	Measure of the Efficiency of the Selected Coding Scheme	19
2.3.1	The Expected Code Length	19
2.3.2	The Data Compression Ratio	20
2.4	The Expected Compression and Expected Saving in File Storage Costs	21

2.4.1 The Savings in File Storage Costs Obtained by Compression of the Titles Field 21

2.4.2 The Savings in File Storage Costs Obtained by Compression of the Series Field 22

2.4.3 The Savings in File Storage Costs Obtained by Compression of the Abstract Field 22

2.4.4 The Total Savings in File Storage Costs Caused by Compression of the Title, Series and Abstract Fields 23

2.5 Factors that Influence the Design of the Dictionary 23

2.6 Theory of Abstract Trees 24

2.6.1 Use of Binary Trees to Support the Selected Coding Scheme 30

2.6.2 Search on the Dictionary 34

2.6.3 Insertion of New Words in the Dictionary 35

III. DICTIONARY DESIGN FOR TITLE WORDS 40

3.0 Introduction 40

3.1 Design of the Dictionary 42

3.1.1 Design of the One Byte Code Table .. 42

3.1.2 Design of the Two Byte Code Table .. 43

3.1.3 Design of the Three Byte Code Table 46

3.2 Updating the Dictionary 47

3.2.1 Method of Updating the Dictionary .. 50

3.3 Method of Decoding 51

3.4 Method of Encoding 55

3.5 Method of Searching on Title Words in the Catalogue Data Base 59

3.5.1 The Document Index for Title Words . 60

3.5.1.1 Design of the Document Index . 60

3.5.1.2 Updating the Document Index .. 62

3.5.1.3 Deletion of Documents from the Document List 64

3.6 Evaluation of Dictionary Design 64

3.6.1 Expected Decoding Time 65

3.6.2 Expected Encoding Time 67

3.6.3 Expected Updating Time 69

3.6.4 Expected Search Time 69

3.6.5 Estimated Size of the Title Word Dictionary 70

3.6.6 Conclusion of Evaluation 71

3.7 Compression of Other Fields of the Cataloque Data Base 72

IV. THE UDC NUMBER DICTIONARY 77

4.0 Introduction 77

4.1 Coding Scheme for UDC Numbers 77

4.2 Design of the UDC Number File 78

4.3 Operations Performed on the UDC Number File 83

4.3.1 Decoding UDC Numbers 83

4.3.2 Encoding UDC Numbers 85

4.3.3 Updating the UDC Number File 85

4.3.4 Searching on UDC Numbers 86

4.4 Design of the Subject Word File 87

4.5 Operations Performed on the Subject File .. 89

4.5.1 Updating the Subject Word File 90

4.5.2 Searching on Subject Words 90

4.6 Translation of UDC Numbers to Subject Words 91

4.7 Translation of Subject Words to UDC Numbers 93

4.8 Amendments to UDC Numbers 93

4.9 Thesaurus for a UDC Based Library 95

4.10 Evaluation of the UDC Number Coding Scheme 97

4.10.1 Disk Storage Saving Attributed to a Compressed UDC Number Field 99

4.11 Evaluation of the Design of the UDC Number Dictionary 100

V. PROGRAMMING CONSIDERATIONS 104

5.0 Introduction 104

5.1 Improved Disk Access Time 107

5.2 Recovery 109

5.2.1 Bulk Tape Saves 109

5.2.2 Audit Trail Tape 110

5.2.3 Disk Scratch Pad 112

5.2.4 Continuous System Operation 112

VI. CONCLUSION 115

BIBLIOGRAPHY 117

APPENDIX 121

LIST of TABLES

Table		Page
1	Decoding Overhead	66
2	Encoding Overhead	68
3	Space Requirement Comparison	73
4	Time Comparison	74
5	UDC Dictionary Overhead	102

LIST of FIGURES

Figure		PAGE
1	Conceptual View of an Integrated On-Line Library System	7
2	Catalogue Search Subsystem	8
3	Classifying Subsystem	11
4	Acquisition Subsystem	13
5	Real-Time Circulation Subsystem	14
6	An Abstract Tree	27
7	A Binary Tree	29
8	One Byte Code Tree	31
9	Two Byte Code Main Tree	32
10	Binary Tree Search Algorithm	36
11	Dictionary Design for One-Byte Codes	44
12	Dictionary Design for Two-Byte Codes	45
13	Dictionary Design for Three-Byte codes	48
14	A Three-Byte Code Dictionary Table	49
15	Dictionary Table before Updating	52
16	Dictionary Table after Updating	53
17	Method of Decoding One-Byte Codes	54
18	Method of Decoding Two-Byte Codes	56
19	Method of Decoding Three-Byte Codes	57
20	Document Index for One-Byte Codes	61
21	Document Index for Two-Byte Codes	63
22	Stable UDC Number Table	80
23	Dynamic UDC Number Table	81

24 UDC-Head 82
25 UDC Number File 84
26 Subject Word File 88
27 UDC Number File 92
28 Thesaurus and its Corresponding UDC Schedules 98
29 General Structure of the On-Line System 106

CHAPTER I

INTRODUCTION

1.0 Introduction

The exponential growth of information is providing most libraries with an unmanageable amount of material to process. In their efforts to survive the rapid and continual change, many libraries are making increasing use of the information storage and retrieval technology that has developed as a result of the introduction of the electronic digital computer.

The field of information storage and retrieval is concerned with methods of creating and managing collections of information in such manner as to facilitate the recovery of pertinent records as they are needed.

1.1 Automation of a UDC Based Library

To date, many on-line library systems have been designed and implemented. Dimsdale [1] gives a detailed review of the more important work that has been done towards the implementation of on-line library automation systems.

With regard to the automation of Universal Decimal Classification (UDC) based libraries, Heaps et al [2], and Freeman and Atherton [3], have shown that the automation of a UDC based library is not only feasible but that the UDC notation lends itself to machine manipulation by computer.

However, very little work has been done to design and implement an economical and efficient on-line-integrated-UDC-based library system.

1.1.1 The Performance of an On-line System

Computers can scan records at speeds that are thousands of times faster than humans can achieve. However it does not always happen that a machine search is faster, more convenient, has as high a recall, or can be as precise as a manual search. For example, a very large file of records in alphabetical order may be stored on a magnetic tape, and a corresponding file may exist either as a card file or as a printed index. To conduct a search for a specific item on the tape by computer might take, on the average, some five minutes, since the tape must be scanned until the item is found. The same item might be found in the manual card file or book index in a few seconds because direct access to the appropriate area is possible and irrelevant material may be skipped very rapidly. Direct access capabilities can also be incorporated into a computer search system, but at present time the storage of very large collections of records on direct access devices may well prove to be prohibitively expensive. A.B. Veaner has stated that 60% of the total cost of operation of the SPIRES-BALLOTS system is attributed to disk storage [4]. The

present charging rate for disk storage by the University of Alberta Computing Services is $1.00 per 4096 bytes per month.

1.1.2 The Construction of a Thesaurus for a UDC-Based Library System

When examining the non-clerical activities of information storage and retrieval systems such as selecting index terms, translating records and evaluating records, it is found that computer techniques have made relatively little impact. A computer will search for a given word with great success, but it will not select a synonymous word unless it has also been given a definition of 'synonym' by an appropriate algorithm. Work has been done in this field by Mercier [5] and Alber [6] in the form of providing a thesaurus to aid the librarian in classifying and the user in searching. However, this approach requires large and expensive computer memories and excessive programming.

1.2 Statement of the Problem

The foregoing sections have referred briefly to the development of on-line UDC based library systems. The design considerations for such systems form the main subject of the present study.

The work reported in this thesis is concerned with two problems. The first problem involves the design, implementation, testing and evaluation of a fast, economical file structure and the related file access strategies for an on-line UDC based library system which can serve both large and small UDC libraries. The second problem involves the design of an economical thesaurus for a UDC based library.

A general discussion of the UDC is given by Harper [7], Miles [8], and the British Standards Institution [9].

The present study was carried out in the Department of Computing Science at the University of Alberta in conjunction with the Library of the Boreal Institute for Northern Studies.

1.3 The Boreal Institute For Northern Studies

The Boreal Institue for Northern Studies was established at the University of Alberta in 1960 to advance '...the acquisition and dissemination of knowledge of the North' [10].

Since that time the Institute has developed into a major research body functioning as an integral part of the University. It also acts as a catalyst and coordinator

between various individuals and organizations with northern research interests.

Research conducted and/or funded by the Institute spans practically all disiplines, and ranges from studies of soils and vegetation to selection of reading materials for Northwest Territory schools. It includes studies relating to the influence of pipelines on the environment.

To aid the research, the Boreal Institute for Northern Studies maintains a library devoted to publications relevant to northern regions.

1.3.1 The Library of the Boreal Institute for Northern Studies

The library of the Boreal Institute for Northern Studies contains approximately 7,000 books and more then 10,000 documents that include reports and periodicals. In addition, it has newspaper clippings and maps. All relate to the circumpolarnorthern countries with particular emphasis on Canada's North and boreal regions.

Because of the specialized nature of the library, it is indexed through the Universal Decimal Classification for Use in Polar Libraries. These schedules are produced by the Scott Polar Research Institute of Cambridge, England. It is a specialized extension of the more general UDC and

is designed to suit the special needs of a polar library as described by Roberts [11,12].

1.4 An Overview of the Automated Library System

The automated system will include a catalogue search subsystem, a computer assisted classifying subsystem, a real-time circulation subsystem, and an acquisition subsystem. Although all these subsystems are designed to operate primarily in in on-line mode, they can also run in batch mode.

Figure 1 gives a highly summarized, conceptual view of the on-line UDC library system. The rectangles in Figure 1 represent those parts of the overall system that are automated. The solid lines represent flow of information between the subsystems and the dotted lines represent flow of physical documents between the subsystems. The following discussion summarizes the functions of the cited subsystems.

1.4.1 The Searching Subsystem

The catalogue search subsystem, illustrated by Figure 2, allows the user to search for books, pamphlets, periodicals, and selected titles of periodical articles. The search may be specified with respect to any combination of accession numbers, title words, authors, publishers, UDC numbers, or subjects. If an item has more than one author,

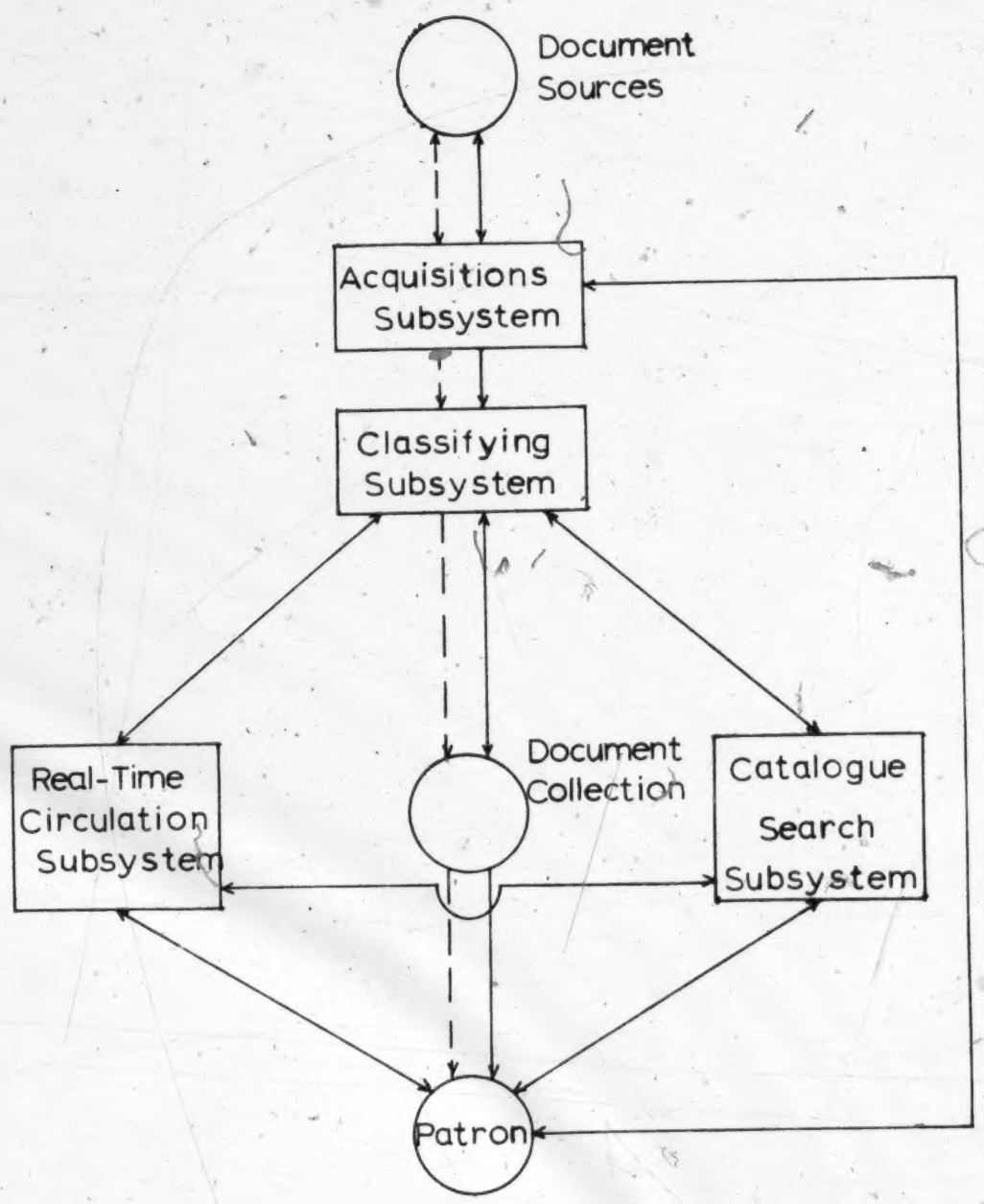

Figure 1
Conceptual View of an Integrated On-Line Library System

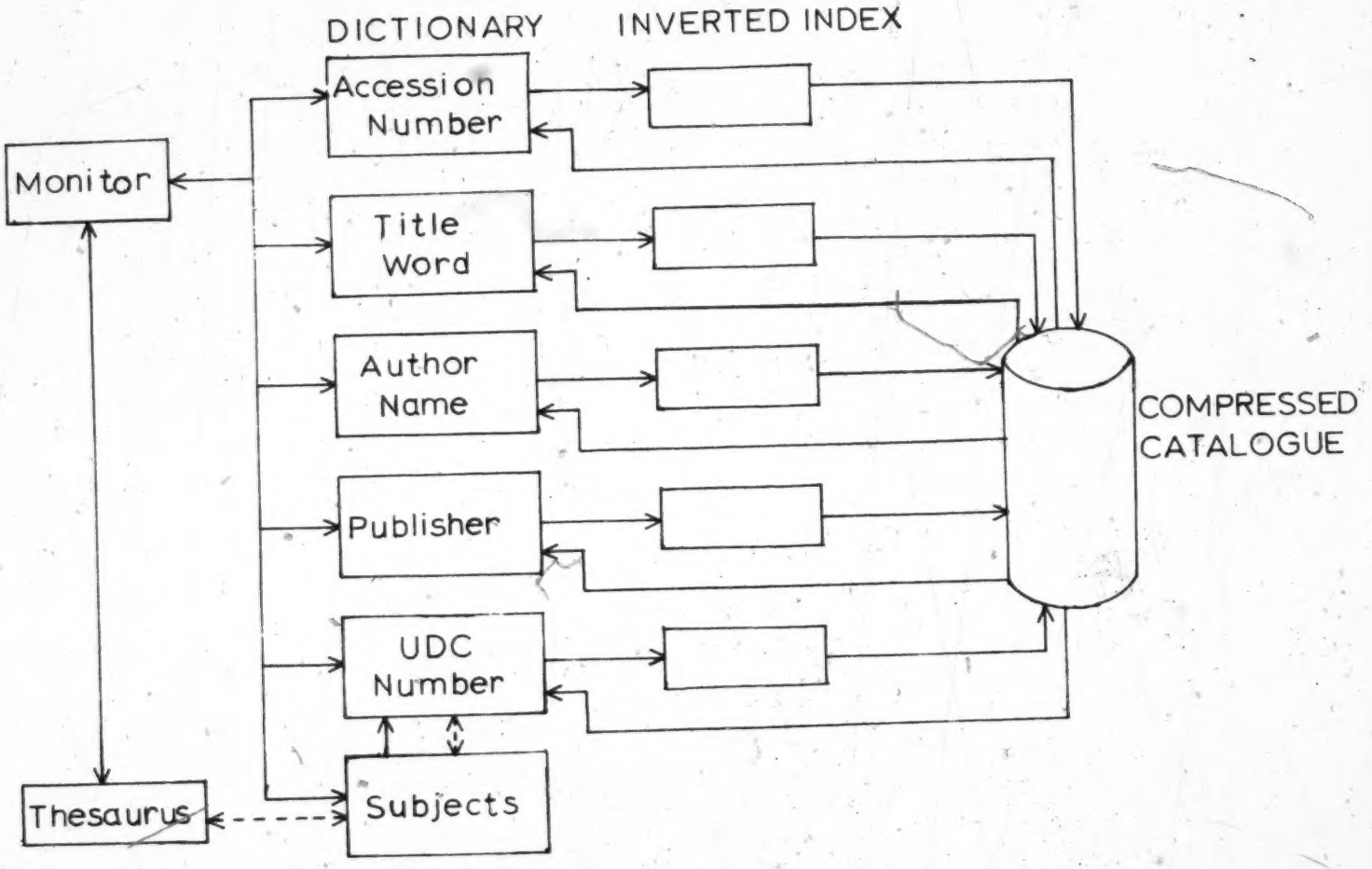

Figure 2
Catalogue Search Subsystem

any single author specification is sufficient to locate the item. Any word or combination of words that appear in the title or describe the subject, will locate the item. If the item has several UDC numbers, then any combination of then will serve to yield the item.

Questions may be expressed using boolean logic operations to connect terms. A precise definition of the query language is given in the Appendix. In this system, each question term may be tagged to indicate whether it is to be searched for as a complete term in the data base, or as the left-most fragment of a larger term.

The design of the data base will support right-truncation to the extent that a search on a right-truncated word requires the the same order of time as a search on an untruncated word.

The composition of the UDC notation combined with the ability to search on right-truncated UDC numbers provides the system with the means to search on units of the classification. Such an ability, not possible to achieve with the LC classification, will allow a search for a group of titles closely related to the desired topic. This ability will also allow the search subsystem to provide a thesaurus directly from the UDC number and subject word dictionaries. The method of obtaining this thesaurus is outlined in Section 4.9.

1.4.2 The Classifying Subsystem

The classifying subsystem requires an interdependent design of the UDC and subject dictionaries. Figure 3 shows the basic structure of the computer assisted classifying subsystem which aids the librarian by performing the following operations:

i) the subsystem will indicate whether a copy of a book being catalogued is already in the library.

ii) given a subject, the subsystem will provide the subject's corresponding UDC number.

iii) given a UDC number, the subsystem will provide the corresponding subjects

iv) the thesaurus can be used to aid in the choice of subjects.

v) the subsystem will permit fast and easily made amendments to UDC numbers and/or subjects.

vi) the subsystem will allow for quick and effortless adding of UDC numbers and subjects. This feature with the previous one will in effect, update the library's version of the UDC schedules.

vii) to aid the librarian in analyzing holdings, the subsystem can produce a frequency count of classification numbers that relate to accessioned titles.

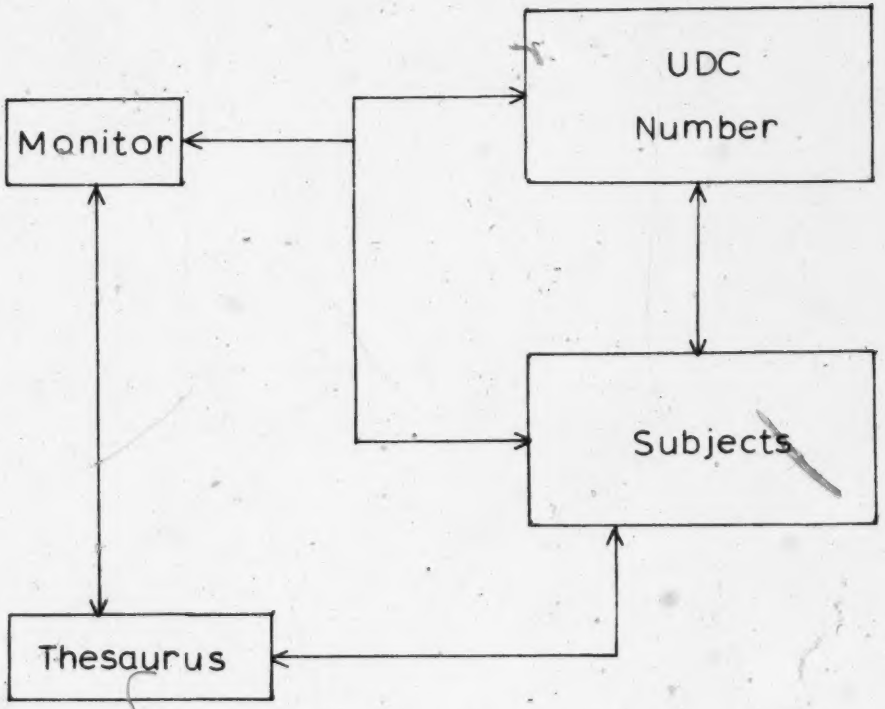

Figure 3
Classifying Subsystem

1.4.3 The Acquisition Subsystem

The basic elements of the acquisition subsystem are outlined in Figure 4. The acquisition subsystem will follow, and account for, the progress of an order from the first request of a title to its physical arrival in the library. This subsystem should have an accounting facility to supply such information as total money spent on books to date, total money reserved for books on order, number of new titles obtained during the past year, total money spent during the past year, and total number of titles in the library.

Once the book arrives, much of the standard processing can be done by the acquisition subsystem. With information provided by the classifying subsystem, the processing will include such operations as assigning accession numbers, producing catalogue cards, and automatically entering the titles into the data base.

1.4.4 The Real Time Circulation Subsystem

The real time circulation subsystem, outlined by Figure 5, provides a complete up to the minute account of all phases of circulation. It will account for titles on loan, titles returned, titles overdue, titles to be renewed, and titles to be reserved. The subsystem also provides a user control file.

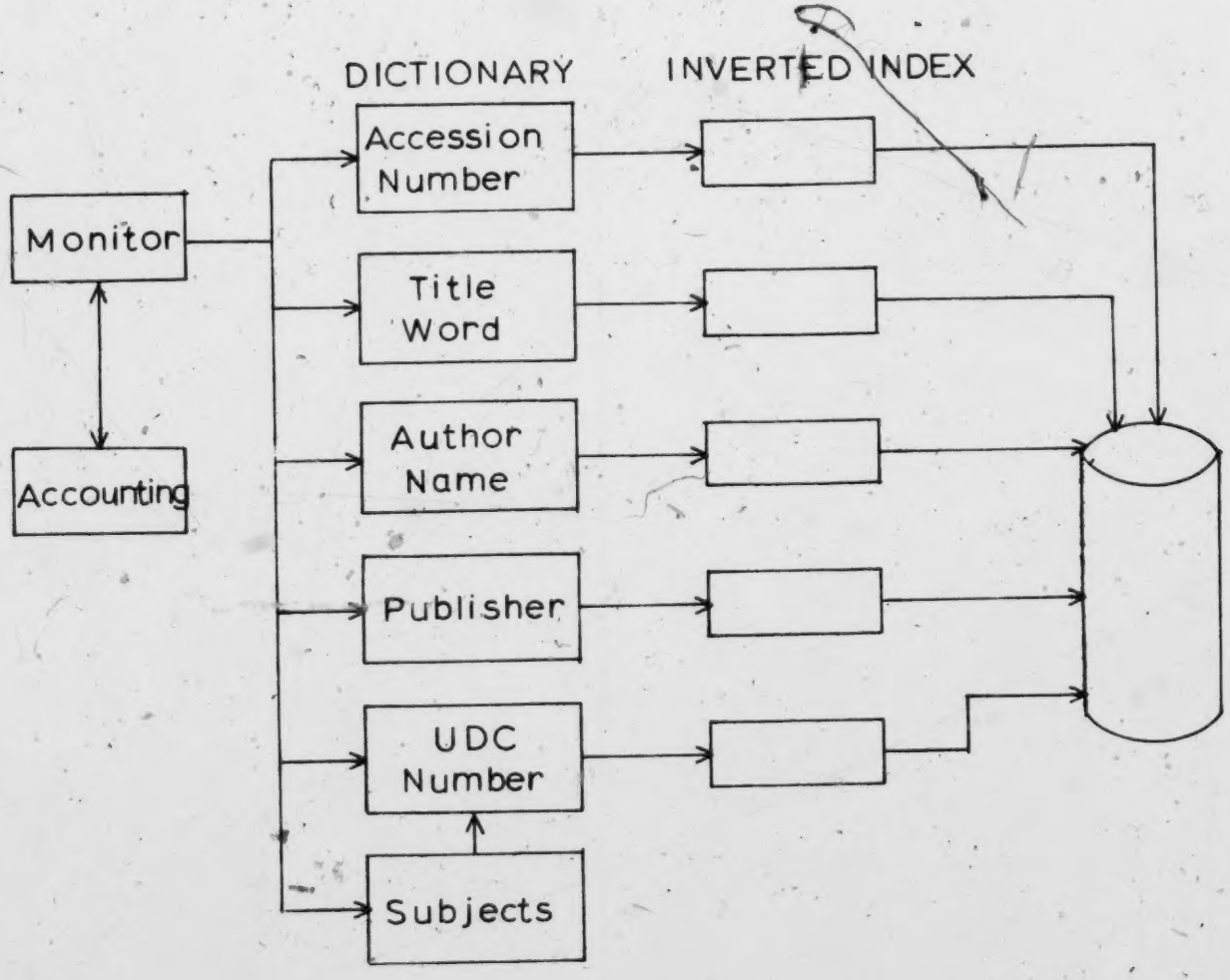

Figure 4
Acquisition Subsystem

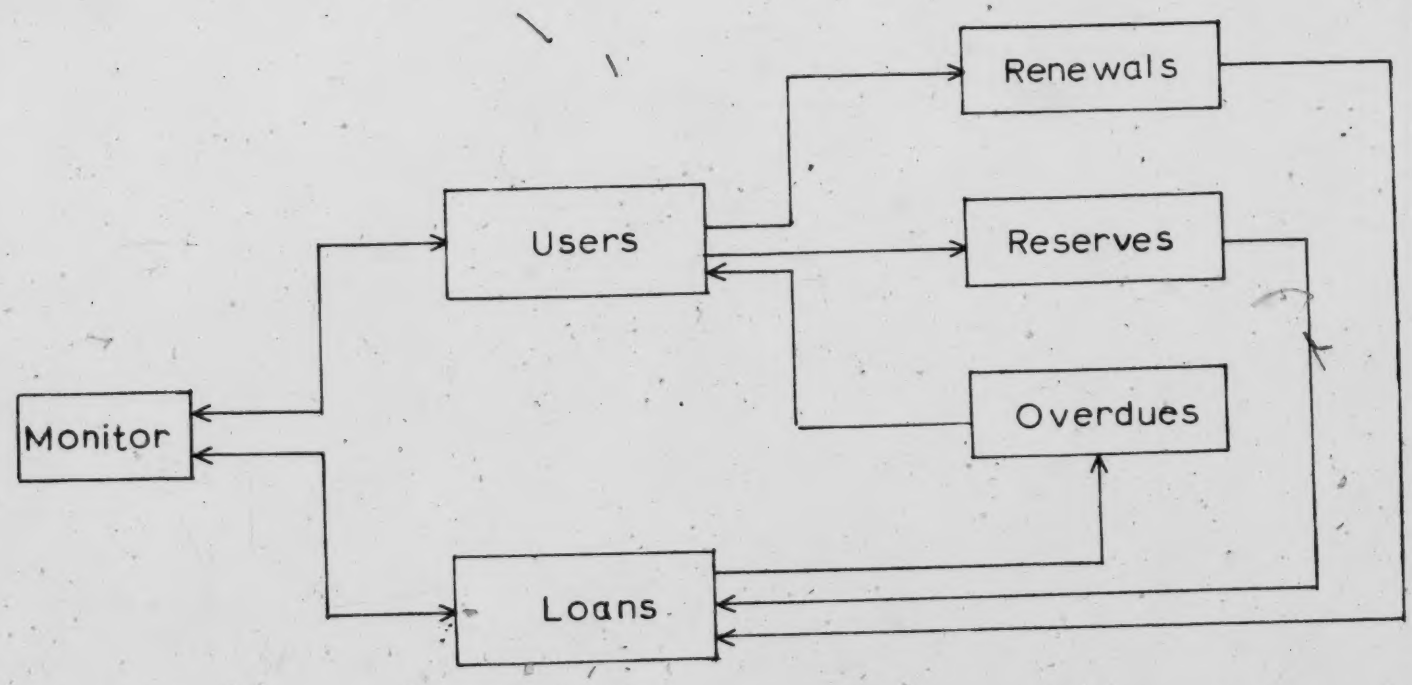

Figure 5
Real-Time Circulation Subsystem

Since the real time circulation subsystem does not directly rely on the compressed coding scheme, the development of the subsystem will not be part of the present study. It should be noted that Dimsdale [1], as part of his masters research designed an excellent circulation subsystem which could be adopted to the system presently being discussed.

CHAPTER II

COMPRESSED DATA BASE

2.0 Introduction

An on-line library system requires the storing of information about the library's holdings on magentic disk. For the proposed system, each record includes informarion about the holding's accession number, call number, author, title, subtitle, publisher, abstract, year, pages, book format, series, bibliography, glossary, L.C. number, I.S.B. number, G.D.C. number, order number, U.D.C. number, and anayltical authors. Each field of the record is named by the contents of the field.

A significant cost of an on-line library system is that required for storage of the holding records on magnetic disk. This study attempts to reduce the required amount of disk storage to an absolute minimum. This reduction is achieved through use of a data base stored in a compressed form.

2.1 The Selection of a Compression Coding Scheme

The coding scheme used for this study was adapted from the scheme proposed by H.S. Heaps and L.H. Thiel [13]. Its basic principle is that the shortest codes are assigned to those words that have the highest frequency of occurrence and the longest codes are assigned to those words that have

the lowest frequency of occurrence. The expected length of the resulting codes is thus significantly less then it would be if all words were mapped into codes of the same length.

This coding scheme was selected over other methods [14,15,16,17,18,19,20,21,22,23,24], because the codes are unique, and hence have no information loss during decoding. In addition, the codes have a very small average length and may be constructed by purely automatic means. The codes are constructed in numerical order; hence all possible codes of a given length may be used before it is necessary to use codes of a greater length.

2.2 A Description of the Selected Coding Scheme

The selected coding scheme as adapted from the scheme proposed by H.S. Heaps and L.H. Thiel [13] is as follows:

i) The most frequent 127 words are coded in one byte as 1XXXXXXX where X denotes 0 or 1. The one-byte code will be the binary representation of a number i such that $128 \leq i \leq 255$. This code will be denoted by (C). It follows from a word-frequency distribution of H. Kucera and W.N. Francis [25] that this code covers 49.572% of all word occurrences in written English text. The 128th code 11111111 will be reserved for use as an escape code.

ii) The next most frequent 16,384 words are coded in two bytes as 0XXXXXXX1XXXXXXX where X denotes 0 or 1. This code will be denoted by (B,C) where the left byte of the code forms the B part of the code and the right byte of the code forms the C part of the code. The B part of the code will be the binary representation of a number i such that $0 \leq i \leq 127$. The C part of the code will be the binary representation of a number j such that $128 \leq j \leq 255$. The (B,C) code will cover another 45.544% of word occurrences in written English text [25].

iii) The next frequent 2,097,152 words are coded in three bytes as 0XXXXXXX0XXXXXXX1XXXXXXX, where X denotes 0 or 1. This code will be denoted by (A,B,C) where the left byte of the code forms the A part of the code, the middle byte of the code forms the B part of the code, and the right byte of the code forms the C part of the code. The A part of the code will be the binary representation of a number i such that $0 \leq i \leq 127$. The B part of the code will be the binary representation of a number j such that $0 \leq j \leq 127$. The C part of the code will be the binary representation of a number k such that $128 \leq k \leq 255$. The (A,B,C) code covers all but an infinitesimal proportion of the remaining 4.884% of word occurrences in written English text [25].

It should be noted that the C part of each code, by being greater than 127, indicates that any immediately following byte on the right is the first byte of a new code.

2.3 Measure of the Efficiency of the Selected Coding Scheme

2.3.1 The Expected Code Length

One measure of the efficiency of a coding technique is the expected code length, $\bar{C}$, which is defined by:

$$\bar{C} = \sum_{j=1}^{M} P(j)C(j), \qquad (2.1)$$

where C(j) is the length of the j-th group of code, $C(j) > C(j-1)$, P(j) is the relative freguency of occurrence of codes of length C(j), and M is the number of different code lengths. In terms of f(i)'s (the relative frequency of occurrence in the material to be coded of the i-th word in the vocabulary) P(j) is defined by:

$$P(j) = \sum_{i=1}^{N(j)} f(i) - \sum_{i=1}^{N(j-1)} f(i), \qquad (2.2)$$

where the N words of the vocabulary are ranked in order of decreasing frequency of occurrence so that $f(i) \leq f(i-1)$ and where N(j) is the total number of words mapped into codes of lengths C(1) through C(j).

For the coding scheme selected for use in the

present study we have:

$$P(1) = 0.49572$$

$$P(2) = 0.45544$$

$$P(3) = 0.04884$$

$$C(1) = 1$$

$$C(2) = 2$$

$$C(3) = 3$$

$$M = 3$$

Using Equation (2.1) we obtain:

$$\bar{c} = \sum_{j=1}^{M} P(j)C(j)$$

$$= P(1)C(1) + P(2)C(2) + P(3)C(3)$$

$$= 0.4957 \times 1 + 0.45544 \times 2 + 0.04884 \times 3$$

$$= 1.55312 \text{ bytes/code.}$$

This expected length is considerably less then that obtained by use of most other schemes.

2.3.2 The Data Compression Ratio

Davidson [26] defines the data compression ratio, Rs, as the ratio of the space required to store the data base if the data base is uncoded to the space required if the data base is compressed by means of a coding technique. The data compression ratio is defined by:

$$Rs = \bar{w} / \bar{c}, \qquad (2.3)$$

where $\bar{w} = \sum_{i=1}^{N} f(i)w(i)$ is the expected length of the word to

be compressed; f(i) is the frequency of the i-th word and w(i) is the length of the i-th word in the vocabulary.

According to Reid and Heaps [27] the expected length of the title word, $\bar{w}$ is 5.6 letters per word. From Section 2.3.1, $\bar{c}$ = 1.55312. Using Equation (2.3) Rs is found to equal 3.67. This implies that the compressed data fields occupy only 27.2% of the space as that is occupied by the same data in a non-compressed form.

2.4 The Expected Compression and Expected Saving in File Storage Costs

2.4.1 The Savings in File Storage Costs Obtained by Compression of the Title Field

According to Reid and Heaps [27] the average number of words per title is 5.5 words. The total space occupied by an uncoded title field of 50,000 titles is

$$5.0 \times 10^4 \times 5.5 \times 5.6 = 1.540 \times 10^6 \text{ bytes.}$$

The space needed by a compressed title field is

$$5.0 \times 10^4 \times 5.5 \times 1.55312 = 4.27 \times 10^5 \text{ bytes.}$$

Thus compression of the title field results in a saving of 1.113×10^6 bytes. The current rate for disk storage by the University of Alberta Computer Services is $1.00 per 4096 byte page per month. Therefore, the compression of the title field represents a saving of

1.113×10^6 bytes x \$1.00/4096 byte-month = \$271.62 per month, per 50,000 titles.

2.4.2 The Saving in File Storage Costs Obtained by Compression of the Series Field

Of the 50,000 titles, close to half have a series field with an average length of ten words. Assuming the average length of each word to be 5.6 characters, the disk space required by the uncoded series fields is

$$2.5 \times 10^4 \times 10 \times 5.6 = 1.4 \times 10^6 \text{ bytes.}$$

The disk space required by the coded series field is

$$2.5 \times 10^4 \times 10 \times 1.55312 = 3.88 \times 10^5 \text{ bytes.}$$

Thus, compression of the series field allows a saving of 1.012×10^6 bytes or 1.012×10^6 bytes x \$1.00/4096 byte-month = \$242.18 per month.

2.4.3 The Savings in File Storage Costs Obtained by Compression of the Abstract Field

The average number of words per abstract is 17. Hence, the space used by a non-compressed abstract field is

$$5.0 \times 10^4 \times 17 \times 5.6 = 4.76 \times 10^6 \text{ bytes,}$$

while the compressed abstract field uses only

$$5.0 \times 10^4 \times 17 \times 1.55312 = 1.32 \times 10^6 \text{ bytes.}$$

This is a saving of 3.44×10^6 bytes or 3.44×10^6 bytes x \$1.00/4096 byte-month = \$839.90 per month.

2.4.4 The Total Savings in File Storage Costs Caused by Compression of the Title, Series, and Abstract Fields

In Sections 2.4.1, 2.4.2 and 2.4.3, it was found that for 50,000 titles a total of 5.565×10^6 bytes of disk storage was saved if the title, series, and abstract fields were compressed according to the scheme described in Section 2.2. Based on the current charge for disk storage by the University of Alberta Computing Services, this represents a saving of $1353.80 per month. Of course, this saving in disk storage cost is partly offset by the costs of encoding and decoding the fields of the compressed catalogue entries. Also, this saving is reduced by the size of the dictionary needed to encode and decode the fields.

2.5 Factors that Influence the Design of the Dictionary

One objective of this study was to design a dictionary that would allow rapid encoding and decoding while at the same time reducing the amount of disk storage needed to store the dictionary to a minimum. If the dictionary can also be used for the searching of records then there is an additional saving of the space that would have been used to store another list of uncoded words needed for the search. This represents an average saving of 280,000 bytes for every 50,000 searchable words.

The theory of abstract trees provides the fundamentals required to design a dictionary which will achieve the above mentioned objectives.

2.6 Theory of Abstract Trees

A tree is a special type of directed graph consisting of elements (nodes) and associations between pairs of elements (branches).

Knuth [28] defines a tree recursively as a finite set T of one or more nodes such that:

a) there is one specially designated node called the root of the tree; and

b) the remaining nodes (excluding the root) are partitioned into $n \geq 0$ disjoint sets $T(1),\ldots,T(n)$ and each of these sets in turn is a tree.

Most of the properties of trees are defined by Salton [29]. The following are the concepts needed to discuss trees in the present treatment.

Path:

If an association is defined between nodes $P(i-1)$ and $P(i)$ for all i, $(i=2,3,\ldots,n)$, a path of length n is said to connect node $P(1)$ to node $P(n)$.

Teriminal nodes:

Nodes P(1) and P(n) are called the terminal nodes of the above path.

Circuit:

A non-trival path whose terminal nodes are identical. A tree is a directed graph which contains no circuits.

Maximal path:

Since a tree contains no circuits, the length of a path in a finite tree is bounded and there exist maximal paths which are not included in any longer paths.

Root:

The initial node of a maximal path.

Leaf:

The final node of a maximal path.

Tree level:

Terminal nodes of paths of length K-1 from a root of the tree are said to lie on the K-th level of the tree. A root lies on the first level of the tree.

Reachable:

If a path exists from node I to node j, node j is said to be reachable from node i.

Subtree:

Any node of a tree defines a subtree of which it is the root, consisting of itself and all nodes

reachable from it. Every subtree of a tree is therefore itself a rooted tree.

Father node:

Each root is said to be the father of the roots of its subtrees. The root of an entire tree has no father.

Son node:

The roots of the subtrees of the same node are said to be brothers or they are sons of the father node.

Degree:

The number of branches leaving a node is called its branching ratio or degree.

Forest:

A set of zero or more disjoint trees form a forest.

These concepts are illustrated in Figure 6 which shows a tree with seven nodes. The root is A, and it has two subtrees {C} and {B,D,E,F,G}. The leafs of the tree are C, D, F, G. The tree {B,D,E,F,G} has node B as its root. Node B is on level 2 with respect to the whole tree, and it has subtrees {D}, {E,G}, {F}; therefore B has a degree indicated left or of 3. E is the only node with a degree of 1; A is the only node with a degree of 2. The maximal path length is 4. Nodes B and C are the sons of node A. Node B is the father of nodes D, E, and F.

The type of tree which supports searching and encoding is a tree of degree two. This type of tree, called

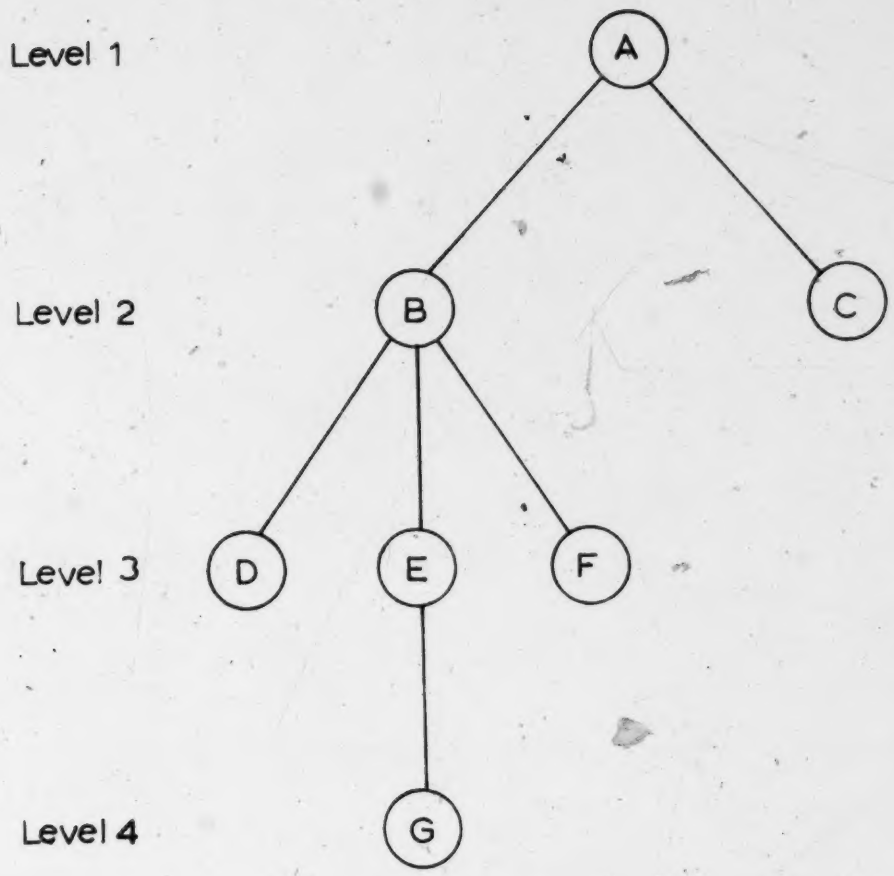

Figure 6

An Abstract Tree

a binary tree, is defined by Knuth [28] as a finite set of nodes which either is empty, or consists of a root and two disjoint binary trees called the left and right subtrees of the root. Figure 7 illustrates a binary tree.

Knuth [28] classifies a binary tree not as a special case of a tree, but as another concept entirely. The basic difference between trees and binary trees is a tree is never empty; and each node of a tree can have 0,1,2,3,...sons. While a binary tree can be empty, and each of its nodes can have 0,1, or 2 sons. In the case of a single son there is a distinction between a "left" son and a "right" son.

For the purpose of this study, a binary tree will be assumed to be the same as an ordered binary tree. In a ordered binary tree the relative order of the nodes is such that the left most node of any binary subtree in the binary tree is the smallest element of that subtree while the right most node of the same subtree is the largest element of the subtree. D < B < H < E < I < A < F < J < C < G is the order for the binary tree in Figure 7.

A two-dimensional binary tree consists of a number of binary subtrees whose roots are joined together to form a binary tree. The result is a main binary tree of the roots with the subtrees hanging down from each node of the main tree. It will be shown in Section 2.6.1 that this type of structure accommodates rapid searching on large binary

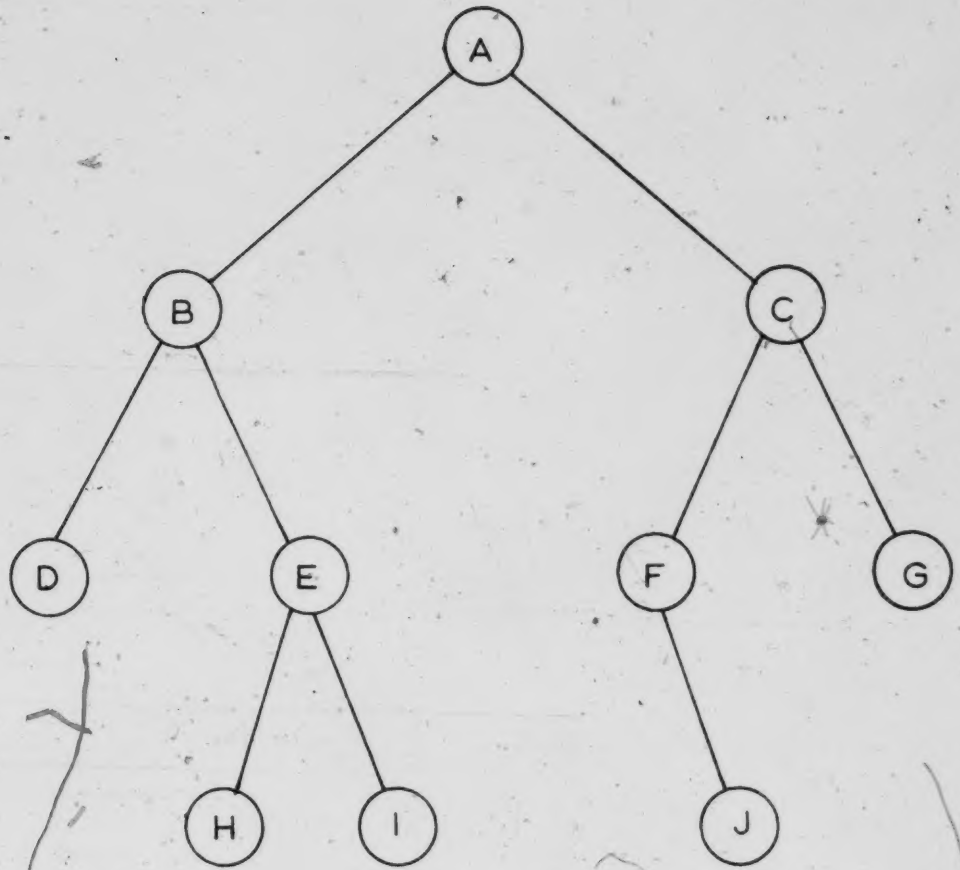

Figure 7

A Binary Tree

trees. Likewise, three-dimensional binary tree consists of a number of two-dimensional binary subtrees whoses roots are joined together to form a binary tree.

2.6.1 Use of Binary Trees to Support the Selected Coding Scheme

The theory of binary trees is used to design the dictionary needed to support the selected coding scheme. This dictionary consists of three main parts, one for each part of the code.

The most frequent 127 words, each of which is coded in one byte, are inserted alphabetically into a binary tree as shown in Figure 8. The next frequent 16,384 words, which are coded in two bytes, are inserted into 128 subtrees of the same design. These 128 subtrees are combined into an alphabetically ordered two dimensional binary tree. The nodes of this tree, illustrated in Figure 9, represent the roots of the 128 subtrees with the corresponding subtree existing in the Z plane below its root.

Likewise, the remaining words which are coded in three bytes are inserted into a forest of 128 two-dimensional binary trees. These 128 two-dimensional trees are combined into an alphabetically ordered three-dimensional binary tree.

Figure 8

One Byte Code Tree

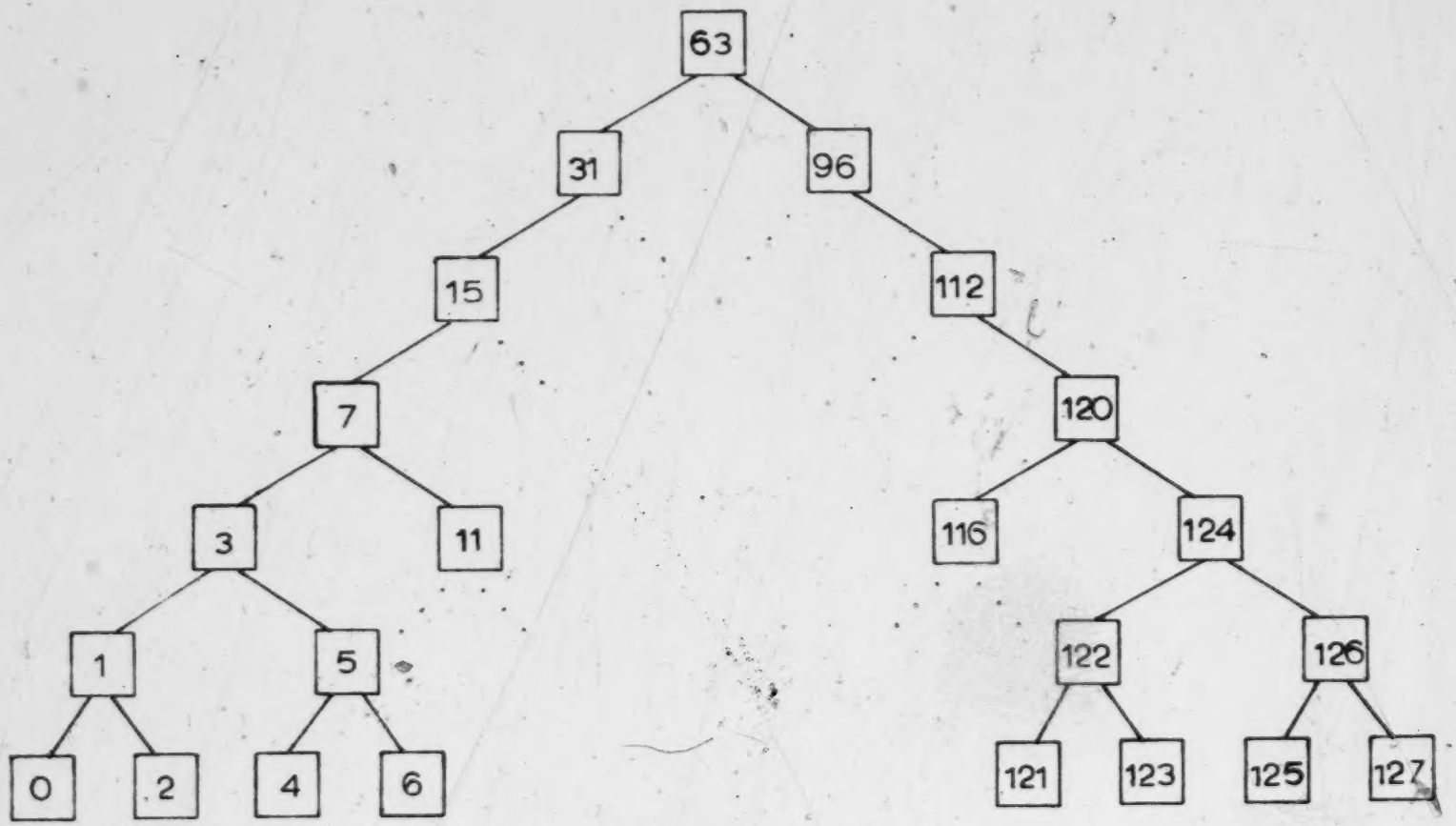

Figure 9

Two Byte Code Main Tree

The code for each word is assigned according to the node it occipies in the tree. The one-byte codes for the 127 most frequent words are established by the following algorithm:

```
        c = 127
        n = root of one byte code tree
A1:     n = left most node of subtree(n)
A2:     c = c+1
        word(n) = c
        n = father(n)
        c = c+1
        word(n) = c
        n = right son(n)
        IF n ≠ leaf GO TO A1
        c = c+1
        word(n) = c
A3:     n = father(n)
        IF n ≠ left son(father(n)) GO TO A3
        n = father(n)
        GO TO A2
```

Using this algorithm the resulting codes are assigned according to the alphabetic order of the words. The numbers at the nodes of the tree in Figure 3 give the code associated with each node.

The two-byte codes are established in the same manner. The algorithm, modified only by initializing c to

-1, is applied to the main tree. The result is that each of the 128 subtrees is assigned a number as given in Figure 9. The original algorithm is then applied to each of the subtrees to obtain the numbering of the nodes of each subtree given in Figure 8. These two numbers give the two byte code of any node; the first byte being the number of the subtree, and the second byte being the number of the node.

Likewise, the nodes for the three-byte code trees are coded. The first byte indicates the two-dimensional subtree of the three-dimensional three-byte code tree which contains the node. The second and third bytes are obtained in the same manner as are the two bytes for the two-byte code.

2.6.2 Search on the Dictionary

The search for a word begins at the root node of the one byte code tree. The desired word is compared to the root node; if the word is equal, the search terminates; if the word is less, the left node of the root is examined; if the word is greater, the right node of the root is examined. This process continues until the desired node is found or it is concluded that the desired word is not in the tree. It is known that the word does not exist in the tree when the

right node does not exist. Figure 10 gives the flowchart for this search algorithm.

If the word is not found, then a similar search is performed on the two-byte code main tree to find the subtree that should contain the desired word. A search for the word is then performed on that subtree. If the word is still not found then searches are made in turn on each two-dimensional subtree of the three-dimensional three-byte code tree until either the word is found or all subtrees have been examined. If the latter is the case, then the word does not appear in the dictionary and it must be inserted into the structure at the last referenced node.

2.6.3 Insertion of New Words in the Dictionary

The binary tree whose search requires, on the average, the examination of the least number of nodes is a balanced binary tree.

A balanced binary tree is a binary tree where the only incompletely occupied level of the tree is the lowest one. Thus two subtrees differ at most by one.

The balance of the tree is maintained by filling the levels of the tree to completion; i.e., no new level shall be initiated as long as vacancies are available in the preceding level.

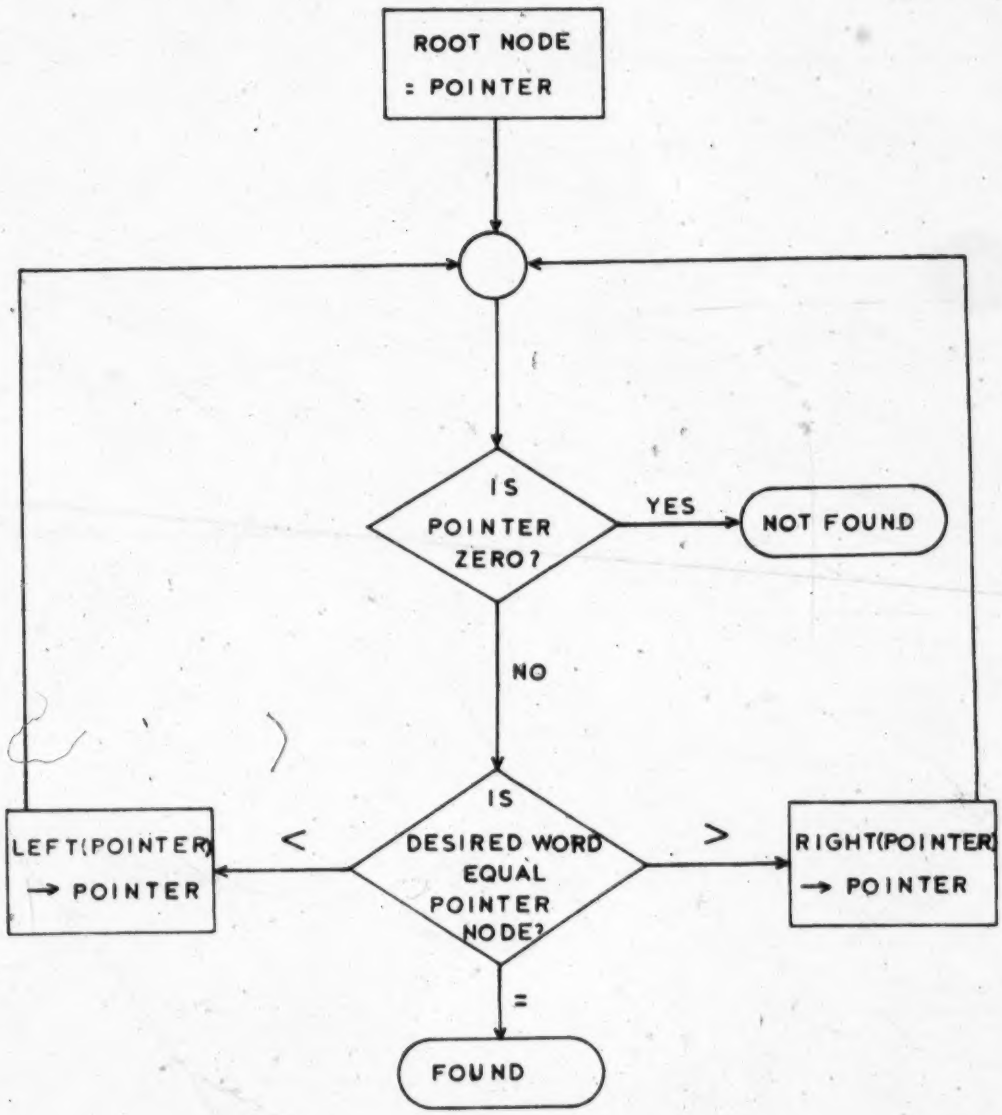

Figure 10

Binary Tree Search Algorithm

In order to follow the balance algorithm of Gauthier and Ponto [30] the following definitions are necessary.

Path nodes:

Nodes on the path designated by symbols (X1,X2,...,Xn) where the root is X1, and the insert node is Xn.

Focal node:

The first path node out of balance.

P-Direction:

For any node Xi in the path, define the path direction at Xi as left if Xi+1 is left of Xi. Path direction is right if Xi+1 is right of Xi.

P-link:

For any node Xi in the path, defined P-link(Xi) as the link pointer in Xi that points to Xi+1.

Q-link:

For any node Xi in the path, define Q-link(xi) as the link pointer in Xi that does not point to Xi+1.

Before applying the balance algorithm, the node to be inserted is first linked to the bottom of the tree.

The balance algorithm must first chart and save the path nodes X1,X2,...,Xn. While doing this it can also determine the focal node Xf. Then, based on conditions about Xf, there are five cases for the algorithm to consider.

Case 1:

IF f=0, THEN

indicator(Xi) = P-direction(Xi) for i=1,2,...,n-1

Case 2:

IF P-direction(Xf) ≠ indicator(Xf), THEN

indicator(Xf) = B

indicator(Xi) = P-direction(Xi) for i=f+1,...,n-1

Case 3:

IF P-direction(Xf) = indicator(Xf) AND

P-direction(Xf) = P-direction(Xf+1) THEN

indicator(Xf) = B

P-link(Xf) = Q-link(Xf+1)

Q-link(Xf+1) = Xf

P-link(Xf-1) = Xf+1

indicator(Xi) = P-direction(Xi) for i=f+2,...,n-1

Case 4:

IF P-direction(Xf) = indicator(Xf) AND

P-direction(Xf) ≠ P-direction(Xf+1) AND

P-direction(Xf) = P-direction(Xf+2) THEN

indicator(Xf) = P-direction(Xf+1)

P-link(Xf+1) = Q-link(Xf+2)

P-link(Xf+1) = Xf+3

Q-link(Xf+2) = Xf

P-link(Xf+2) = Xf+1

P-link(Xf-1) = Xf+2

indicator(Xi) = P-direction(Xi) for i=f+3,...,n-1

Case 5:

IF P-direction(Xf) = indicator(Xf) AND
P-direction(Xf) ≠ P-direction(Xf+1) AND
P-direction(Xf) ≠ P-direction(Xf+2) THEN
indicator(Xf) = B
indicator(Xf+1) = P-direction(Xf)
P-link(Xf) = Xf+3
P-link(Xf+2) = Q-link(Xf+2)
P-link(Xf+2) = Xf
Q-link(Xf+2) = Xf+1
P-link(Xf-1) = Xf+2
indicator(Xi) = P-direction(Xi) for i=f+3,...,n-1

It is easily seen that this process of inserting new words in the dictionary is rather complex and will require a non-negligibile amount of C.P.U. time for its execution. However, it should be noted, that a word is incorporated in the dictionary only once, although during its lifetime in the dictionary it may be searched and retrieved many times. Consequently, in the organization of an information retrieval system, the emphasis should be on the retrieval operations whereas the efficiency of the efficiency of updating is of secondary importance.

CHAPTER III

DICTIONARY DESIGN FOR TITLE WORDS

3.0 Introduction

For every on-line information system that uses a compressed data base, the dictionary is the component that most directly influences the efficient performance of the system.

The time taken to decode a record is of prime importance. Since all records are stored in compressed form, then each time a record is retrieved for output it must first be decoded. Without efficient decoding, the coding overhead would soon counterbalance the saving in disk storage produced by use of a compressed data base.

For similar reasons, the design of the dictionary must allow rapid encoding and searching. In addition, the dictionary is more powerful if it is structured to support rapid right-truncation searching.

To keep the cost of storing the dictionary on disk low, the design of the dictionary is such as to include only those fields which are necessary to preform the various dictionary operations. As a result, the length of each word and the left and right pointers of each node were eliminated. They were replaced by the relative address of each node stored at the top of each dictionary table. The use of the relative node address with a binary search

provide implied left and right pointers, while the subtraction of two adjacent address provides the length of word at the corresponding node.

The resulting design appears to be more of a modified list structure then a tree structure. However the result of the searching on the dictionary tables is the same as if they were in the conventional tree form of including the left and right pointers with each each node of the binary tree.

The disadvantage of the chosen storage technique is that updating of the dictionary, described in Section 3.2.1, is more involved. However the constant cost of the extra storage out weights the small amount of extra processing time required by the infrequent updates.

Even this reduced size of the dictionary prohibits residence of the entire dictionary in core, the greatest factor that influences the speed of the above operations is the time required to access enough of the dictionary file to perform the specified operation. This chapter will present and evaluate a dictionary which is designed to reduce operational time to a minimum.

3.1 Design of the Dictionary.

The dictionary will be divided into three sections, one for each length of compressed code.

3.1.1 Design of the One-Byte Code Table

To construct the dictionary, the words that appear in the abstract series and title fields of the library holdings are arranged according to their frequency of occurrence. The 127 most frequent words are arranged in alphabetic order and inserted in a table, DIC(0), starting at the 260-th byte of the table.

In the same order, the address of the beginning of each word is inserted in table DIC(0), starting at the first byte. This two byte address is relative to the 260-th byte of the table. Hence the i-th address plus 260 will give the location of the start of the ith word in DIC(0). Likewise the i+1-th address minus the i-th address will give the length of the i-th word. Thus the 128-th address is needed to indicate the length of the 127-th word.

The code for a word that appears in DIC(0) is the binary representation of the position of the term in DIC(0) plus the binary number that represents 127. In this manner we obtain the one-byte binary code (C) such that $128 \leq C \leq 255$ as described in Section 2.2.

Figure 11 illustrates the design of table DIC(0), where the i-th address is noted as A(i) and the i-th word is noted as W(i).

3.1.2 Design of the Two-Byte Code Table

The next 16,384 most frequent terms are alphabetized and then split into 128 lists of 128 words. The lists are inserted in order into tables DIC(n), $1 \leq n \leq 128$, in the same manner as described in Section 3.1.1.

A header table HEAD(0) is created for the set of tables by placing in order the first four characters of the first word of each table into HEAD(0). Likewise, the four byte relative disk location of each table is placed in order into HEAD(0). This header table is used to obtain direct access of the two byte code tables.

The set of two byte code tables, DIC(n) $1 \leq n \leq 128$, and header table, HEAD(0), are illustrated in Figure 12. The i-th address A(i) of the j-th table plus 260 gives the location of the i-th word, W(i), in the j-th table, DIC(j). The four characters of C(j) are the first four characters of the first word in table DIC(j) and the address P(j) gives the relative disk location of table P(j).

Each word that appears in this set of tables will be coded by a two-byte code (B,C) as described in Section 2.2. The first byte is given by the binary representation of the

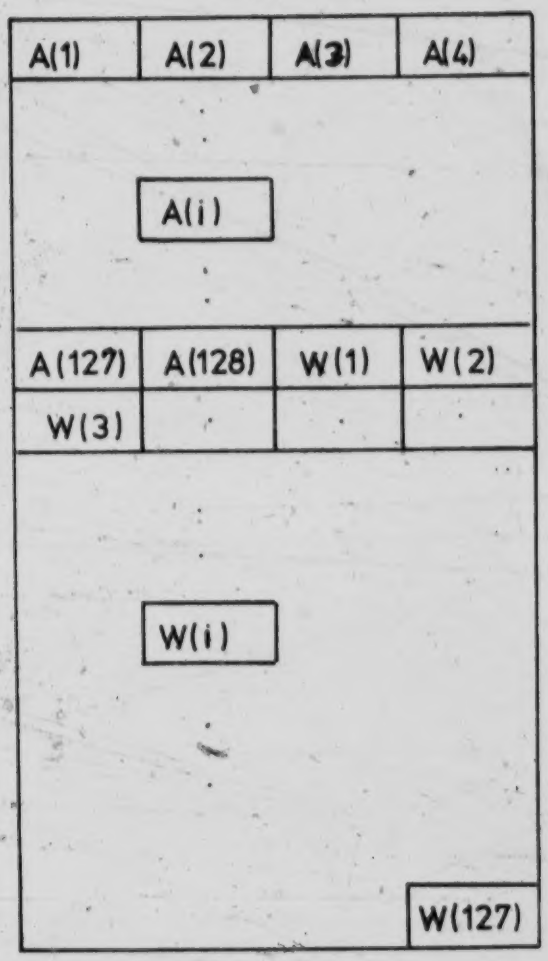

Figure 11

Dictionary Design for One-Byte Codes

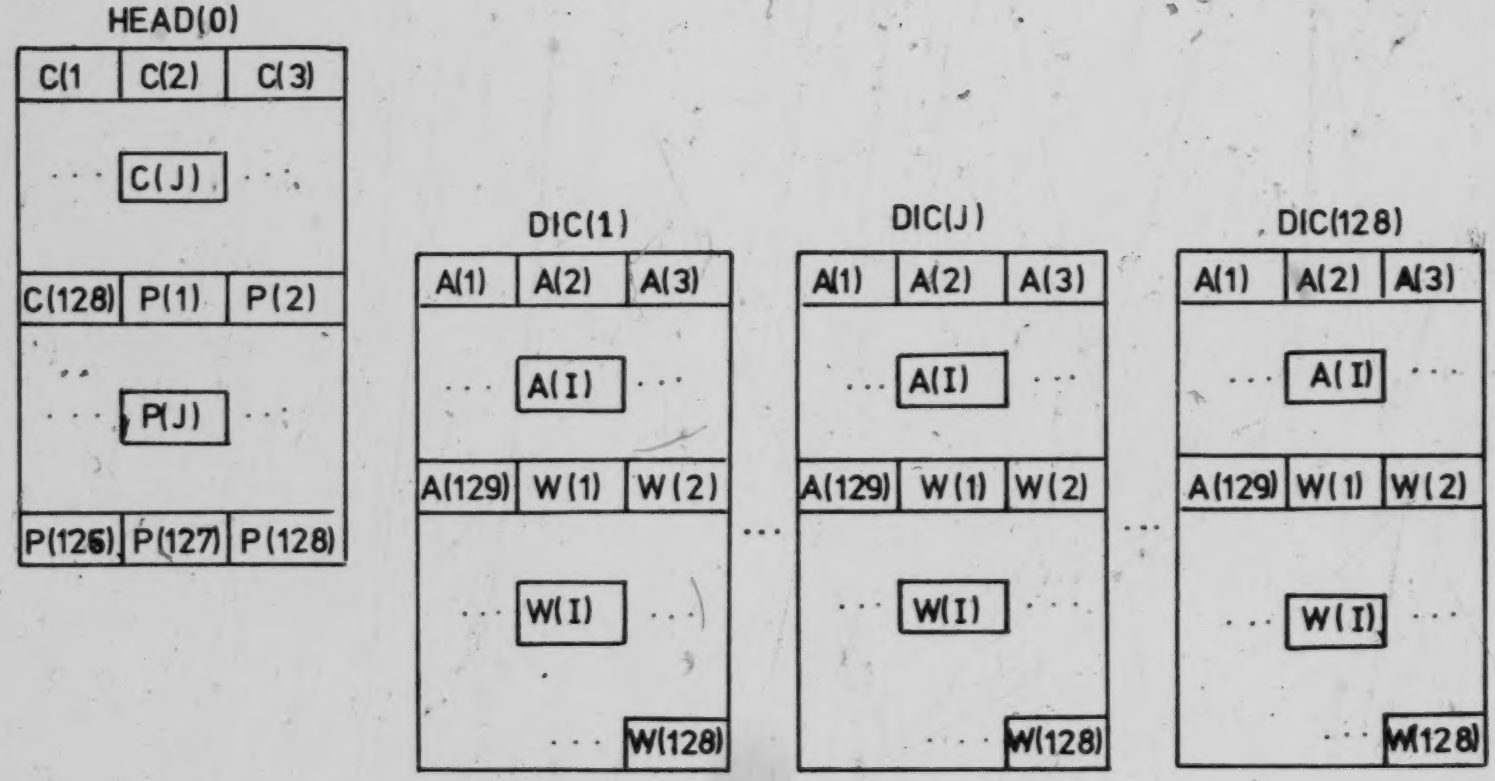

Figure 12
Dictionary Design for Two-Byte Codes

table that contains the term. The second byte is the binary representation of the position the term occupies in that table plus 127.

3.1.3 Design of the Three-Byte Code Table

If there are more than 16,384 words remaining, proceed as in Section 3.1.2, setting up sets of 128 tables until fewer than 16,384 words remain. The codes for the words in these sets of tables will be the three byte codes (A,B,C) as described in Section 2.2. The C byte of the code is given by the set of tables in which the word appears; the B and C bytes are obtained in the same manner as the B and C bytes of the two byte code.

When fewer than 16,384 words remain, the previously filled sets of 128 tables are examined to obtain a 128 part division of the alphabet according to the word usage. Using these divisions as a guide, the remaining words are inserted alphabetically into the next set of 128 tables.

Since these tables will not be full, when a new word is added to the dictionary it will be added to this set of tables. Because the operations of searching and encoding require alphabetically ordered tables, updating these tables could change the position of a word in a table. Hence the position of the uncoded word cannot be used as a constant indication of the C part of the code, (A,B,C). Instead the

number of words, that are in the table when a word is entered, plus 128 will be used as the last byte of the code. To allow encoding and decoding, an indication of this part of the code must also be stored in the table.

When the tables are constructed, the address and word are inserted as outlined in Section 3.1.1 and 3.1.2 except that the words start at the 388-th byte and that the addresses are relative to 388 instead of to 260. The one byte indicators of the third byte of each code are stored starting at byte 260. Figure 13 illustrates this group of tables and Figure 14 illustrates the i-th table of the j-th set.

To obtain direct access to these tables a set of header tables, HEAD(n) where $1 \leq n \leq 127$, are designed similar to HEAD(0) as described in Section 3.1.2. The i-th set of four characters, C(i), of the j-th header table, HEAD(j) are the first four characters of the first word in the i-th table of the j-th set of dictionary tables. Similarly the i-th address, P(i), of HEAD(j) will give the relative disk location of the i-th table of the j-th set of dictionary tables.

3.2 Updating the Dictionary

As a new record is added to the catalogue data base, there is a remote possibility that it contains at least one

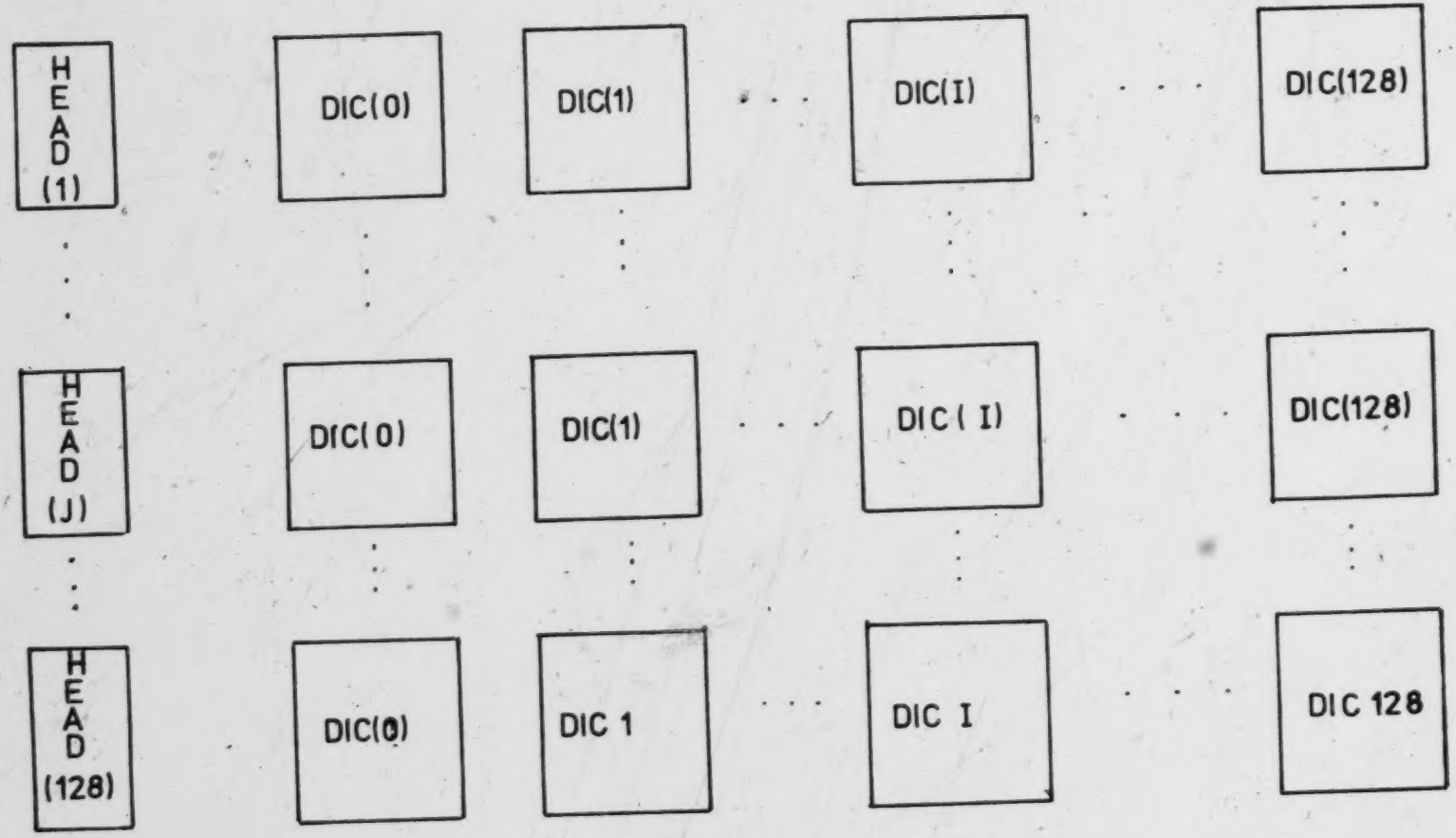

Figure 13

Dictionary Design for Three-Byte Codes

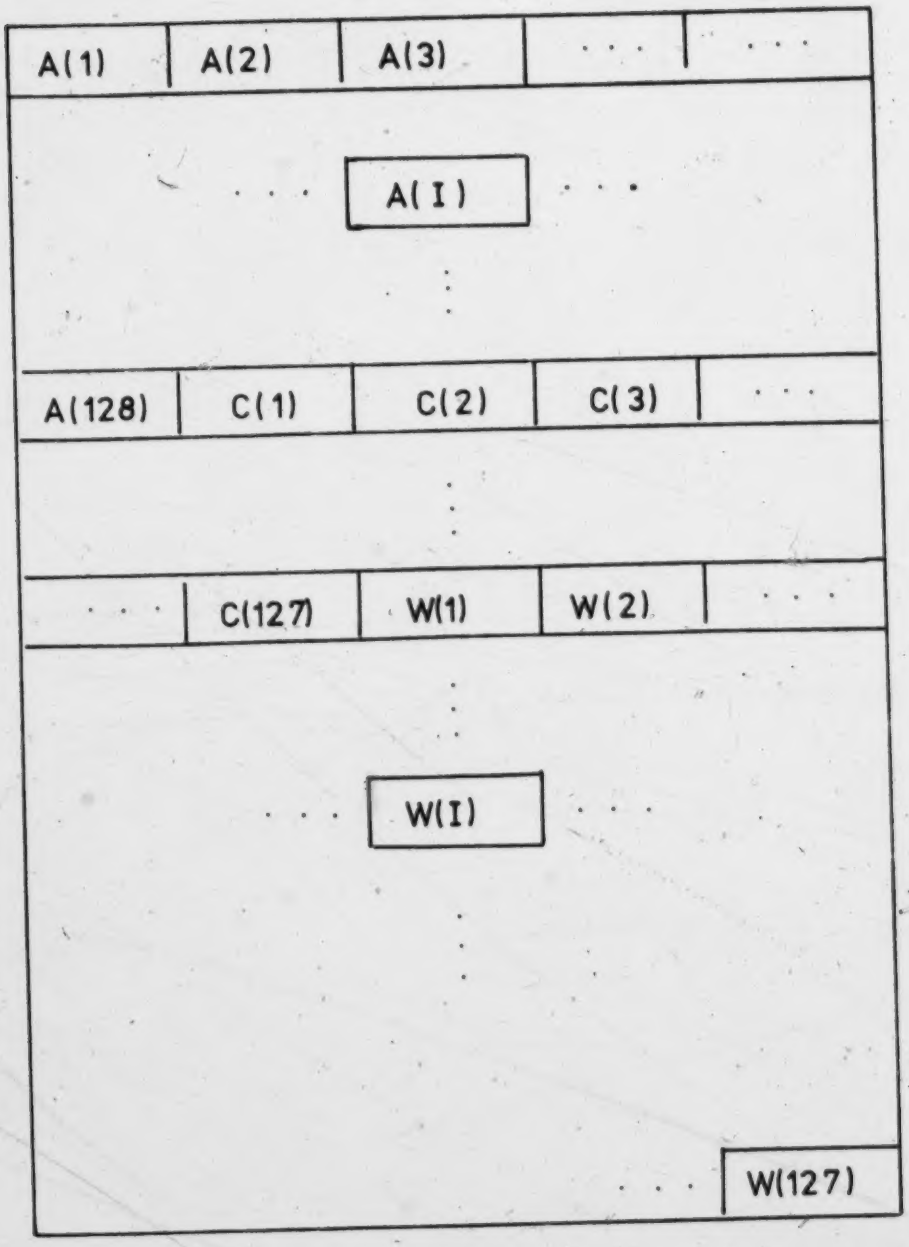

Figure 14

A Three-Byte Code Dictionary Table

word that does not appear in the dictionary. In this case, to allow a completely compressed data base, and to allow searching on all words, all new words must be added to the dictionary. The new words will be added to the last constructed set of dynamic dictionary tables.

3.2.1 Method of Updating the Dictionary

A new word is inserted into the dictionary at the location found by the following method. A binary search is preformed on the header table of the latest set of dynamic tables. A check is made of the number of words already in the corresponding dynamic table. If the table already contains 128 words, then this set of dictionary tables is considered complete and a new set is constructed. In the construction of the new set of tables the previously defined 128 divisions of the alphabet are refined with respect to the word distribution of the last set of tables. Using the same method as described in Section 3.1.3, the new word is inserted into the new set of dictionary tables.

If the table contains fewer then 128 words, then a binary search is preformed on the table to find the position where the new word is to appear. All words in this table, from this position on, are in turn shifted the length of the new word down the table. The new word is then inserted into the table. Also the addresses are updated relative to the new word positions.

Each convert code that is equal to, or greater than, the position of the new word is increased by one. Likewise, the position of the new word is inserted into the convert code postion determined by the number of words in the table.

Figure 15 shows DIC(i), which contains n words, $n < 128$, before a new word W(k) of length l is inserted in postion h. Figure 16 illustrates DIC(i) after W(k) is inserted into position h, where:

i) $A'(m) = A(m)$ for $0 \leq m \leq h$ and

$A'(m) = A(m-1)+1$ for $h < m \leq n+1$

for all m, such that $0 \leq m \leq n$

ii) $C'(m) = C(m)$ for $C(m) < j$ and

$C'(m) = C(m)+1$ for $C(m) > h$

$C'(n+1) = n$

3.3 Method of Decoding

To decode a record in the compressed data base, each code is extracted in turn from the record.

To decode codes of one-byte, 127 is subtracted from the code and the result is multiplied by two to give the location of the relative address of the desired word in table DIC(0). This address plus 260 gives the start of the word that corresponds to the code. Figure 17 illustrates this process.

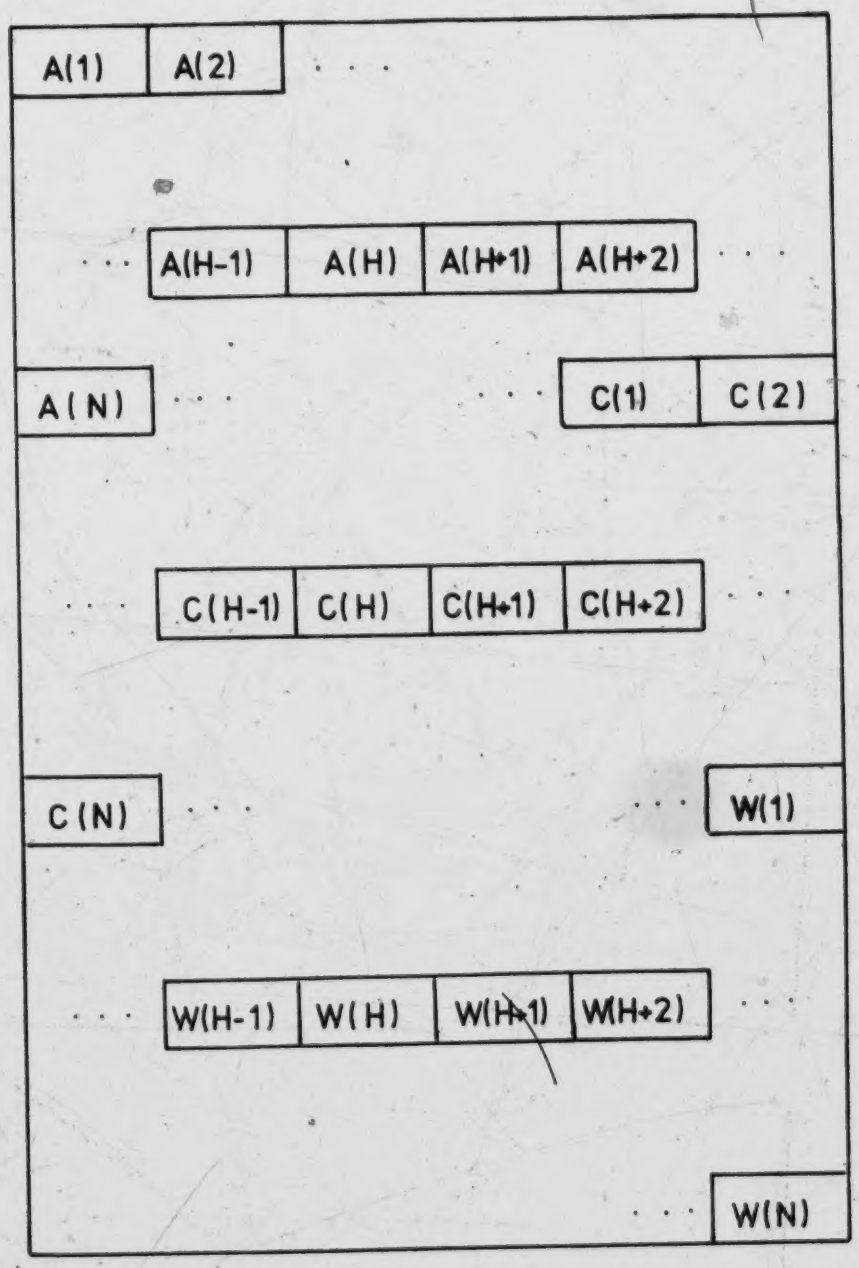

Figure 15

Dictionary Table before Updating

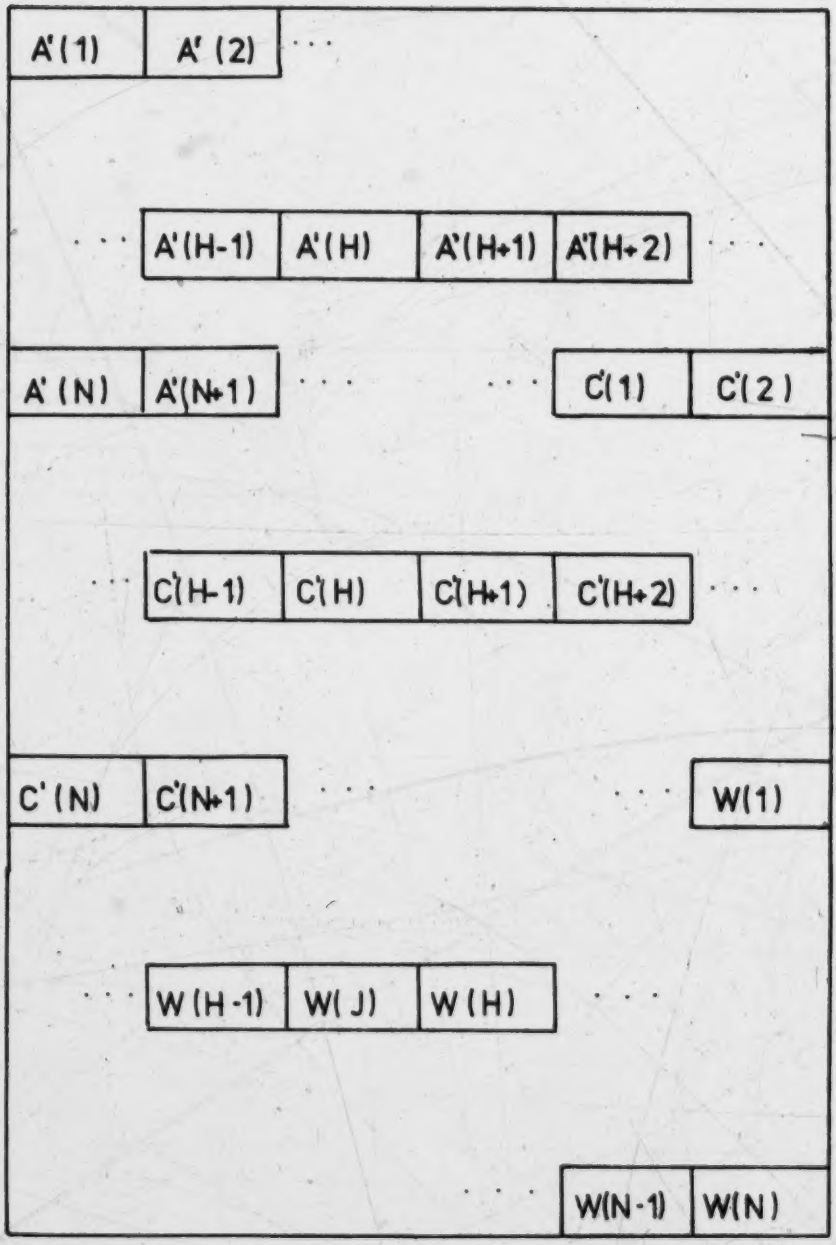

Figure 16

Dictionary Table after Updating

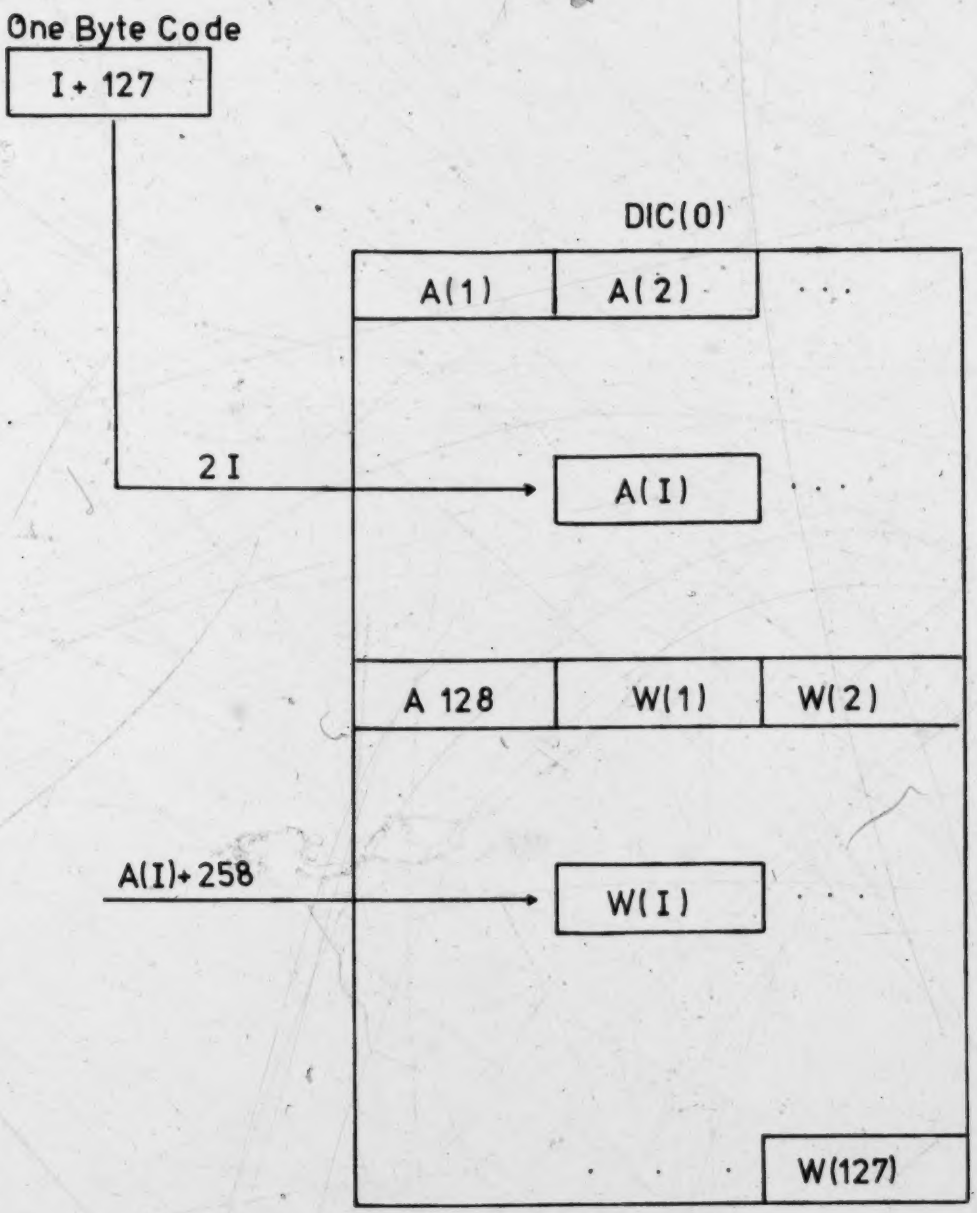

Figure 17

Method of Decoding One-Byte Codes

To decode two-byte codes, the first byte is multiplied by four and the result is increased by 512 to give the relative disk address in HEAD(0) of the table that contains the desired word. This table is read into core. Using the second byte of the code, the desired word is found in the same manner as described for the one-byte codes. The decoding of two-byte codes is illustrated by Figure 18.

To decode a three-byte code, use the fact that the first byte indicates in which set of tables the word appears. The header table HEAD(k) for that set of tables is therefore read into core. Four times the second byte of the code plus 512 yields the disk address of the dynamic table, DIC(j). The last byte of the code less 127 plus 260 gives the location of the convert code. Twice the value of this convert code gives the relative address of the desired word. This address plus 388 gives the starting location of the desired word. Figure 19 illustrates the decoding of a three byte code.

3.4 Method of Encoding

Each time a new title is added to the catalogue data base, each word in the title, abstract, and series field must be encoded to produce a compressed data base.

For each word to be encoded, a binary search for the word is performed on DIC(0). If the word is found in

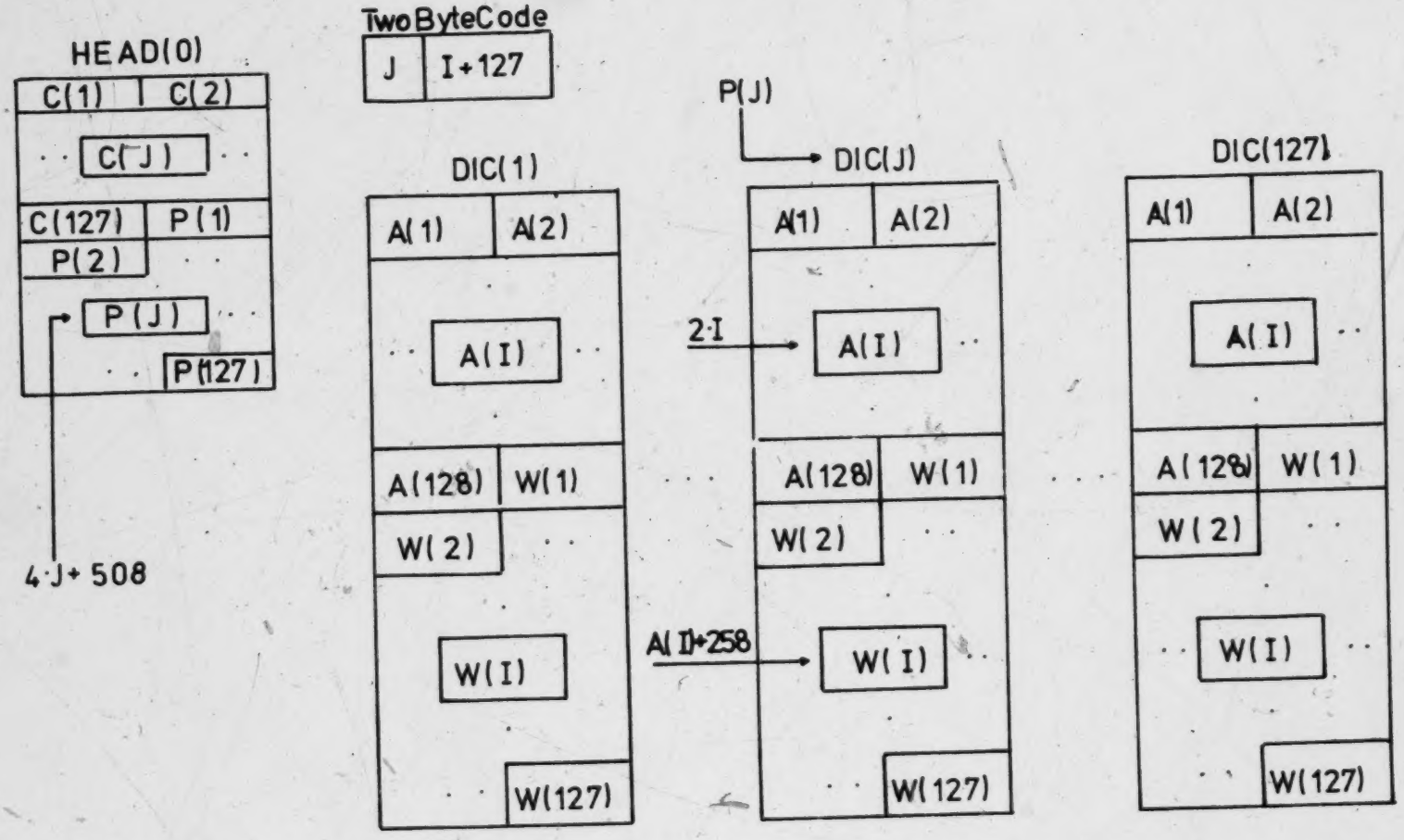

Figure 18
Method of Decoding Two-Byte Codes

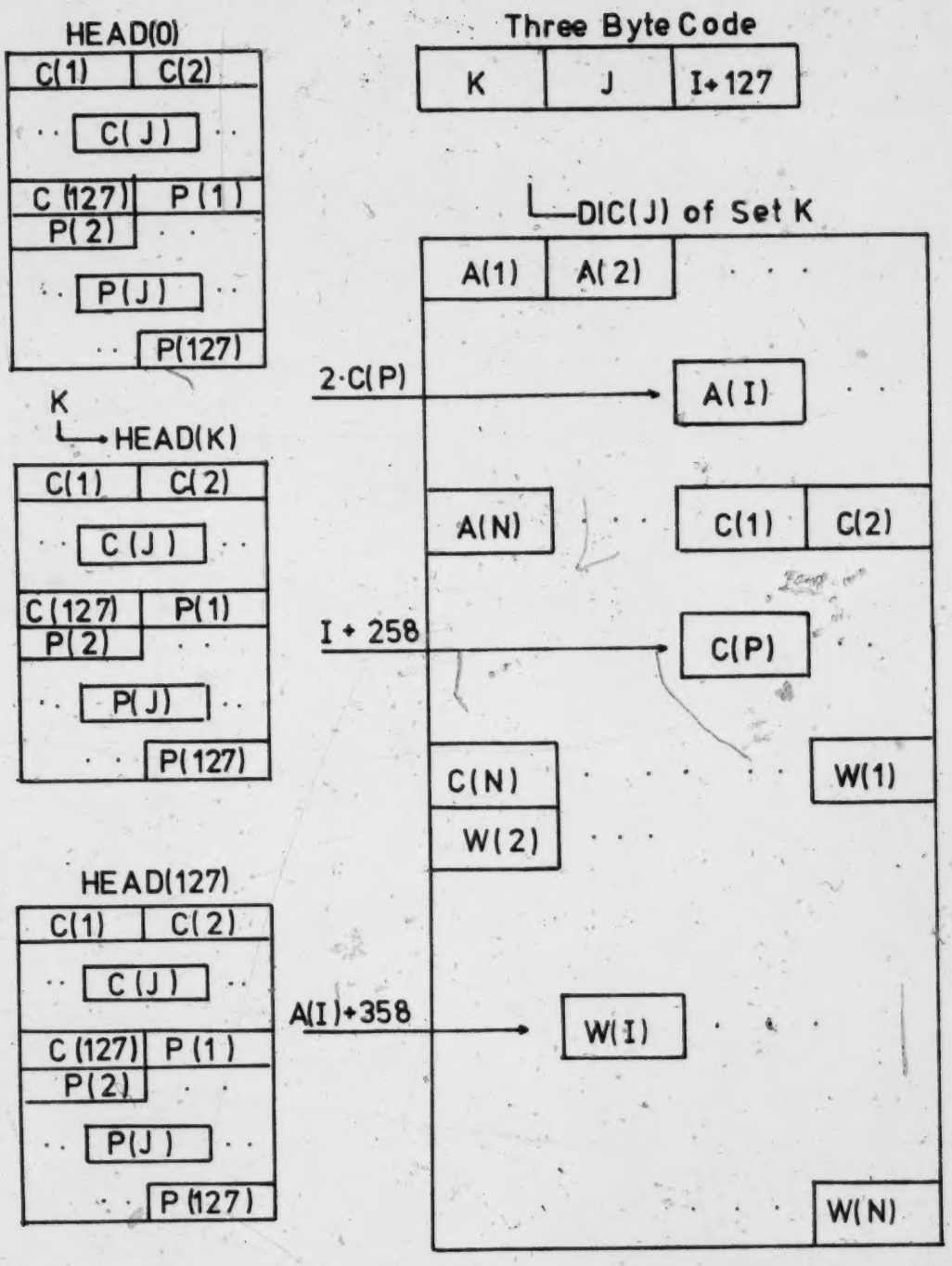

Figure 19
Method of Decoding Three-Byte Code

DIC(0), then the position of this word in DIC(0) plus 127 is the one-byte code for this term.

If the word is not found in DIC(0), then a binary search on HEAD(0) finds the two-byte table that should contain the term. A binary search for the word is performed on DIC(n). If the word is found in DIC(n), then that word is coded in two bytes. The table number will form the first byte of the code and the position of the word in the table plus 127 will form the second byte of the code.

If the word is not found, a binary search is performed in turn on the header tables HEAD(m), $1 \leq m \leq 128$. After each HEAD(m) is searched, the appropriate three byte table is read into core and searched. This process is continued until either the word is found or all sets of tables have failed to provide the word. If the latter is the case, then the word is not in the dictionary.

These words are coded in three bytes. The first byte will indicated the set of tables that contains the word. The number of the dictionary table that contains the word forms the second byte of the code. A linear search through the convert codes of DIC(n) is made to find the pointer to the position of the desired word in DIC(n). The position of this pointer plus 127 gives the third byte of the code.

3.5 Method of Searching on Title Words in the Catalogue Data Base

Search queries are formulated according to the syntax defined in the Appendix. For each question entered, the system interprets the query and retrieves the document list for each query word. It then performs the stated search logic on these document lists. All the records that are referenced by the resultant document list are retrieved, decoded, and printed out.

The document list for a query word is found from the compressed codes of the query word. The codes serve as pointers into the Document Index to the appropriate list.

The important feature of the dictionary structure is that it allows for fast searching on right truncated words, with a minimal increase in storage and disk access time. A right truncated search is a search for all words that begin with a given word fragment.

For a search specified in terms of right truncation, the system searches DIC(0) for the given root. If the root is in DIC(0) then the code for the least alphabetic word that starts with that root is returned. Since the words are in alphabetic order, a linear search down the table returns the codes of all the words in DIC(0) that have that starting root. Note that words with this starting root could appear in all the sets of tables. Hence the above type of search

must be executed on all sets of tables in the dictionary.

After all the tables are searched, the document lists for the words that satisfy the truncation are "OR"ed, the resultant list giving all documents which satisfy the truncation.

3.5.1 The Document Index for Title Words

To support the retrieval of document lists according to compressed codes, the Document Index must be subdivided into three sections, one for each size of code. The design of the index was also influenced by the necessity of on-line amendments to the index.

3.5.1.1 Design of the Document Index

When the system is created, the keys to all the documents that can be retrie by the words in DIC(0) are entered into INDEX(0). The design for the one byte code index is illustrated in Figure 20. The i-th list, L(i), contains the keys of all the documents that are referred to by word W(i) in DIC(0). The keys in L(i) are in ascending numerical order. Immediately following the last key in L(i) is a pointer, Pt, initially set to zero. This pointer will be used to indicate the continuation of the list of keys created by the addition of records to the data base. Stored after DINDEX(0) is a dynamic table to allow the addition of

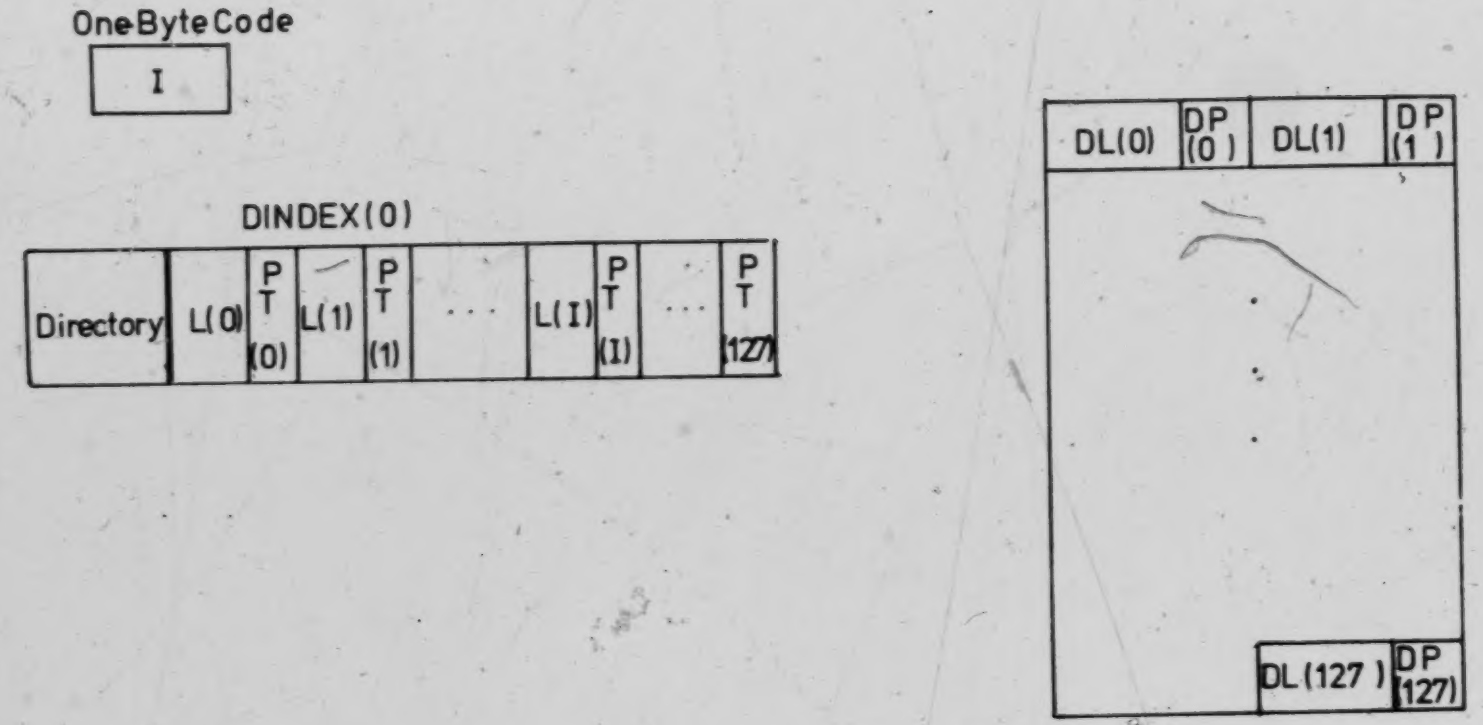

Figure 20

Document Index for One-Byte Codes

to allow the addition of data base. The dynamic table is divided into fixed length list, DL(j) to hold new keys each DL(j) ends with a continuation pointer DP(j) initially set to zero. The size of the dynamic table and the length of each division is dependent on the frequency at which new items are added to the library.

An additional 128 indexes, DINDEX(n), $1 \leq N \leq 128$, are established for the two byte codes. Figure 21 illustrates this set of tables. One-hundred-and-twenty-eight of these sets of tables are created for the three byte codes.

3.5.1.2 Updating the Document Index

For every searchable word in a new record being added to this catalogue data base, the record's storage key must be added to the searchable word's corresponding document list DL(i).

If the pointer PT(i) is zero a new list is started with the new key stored at the next location in the dynamic table. PT(i) is set to point to this list. If PT(i) is not zero, the dynamic list DL(j) indicated by PT(i) is retrieved.

If DL(j) is not full then the new key is inserted into the next empty position in DL(j). If DL(j) is full, the contents of pointer DP(j) governs the next step. The

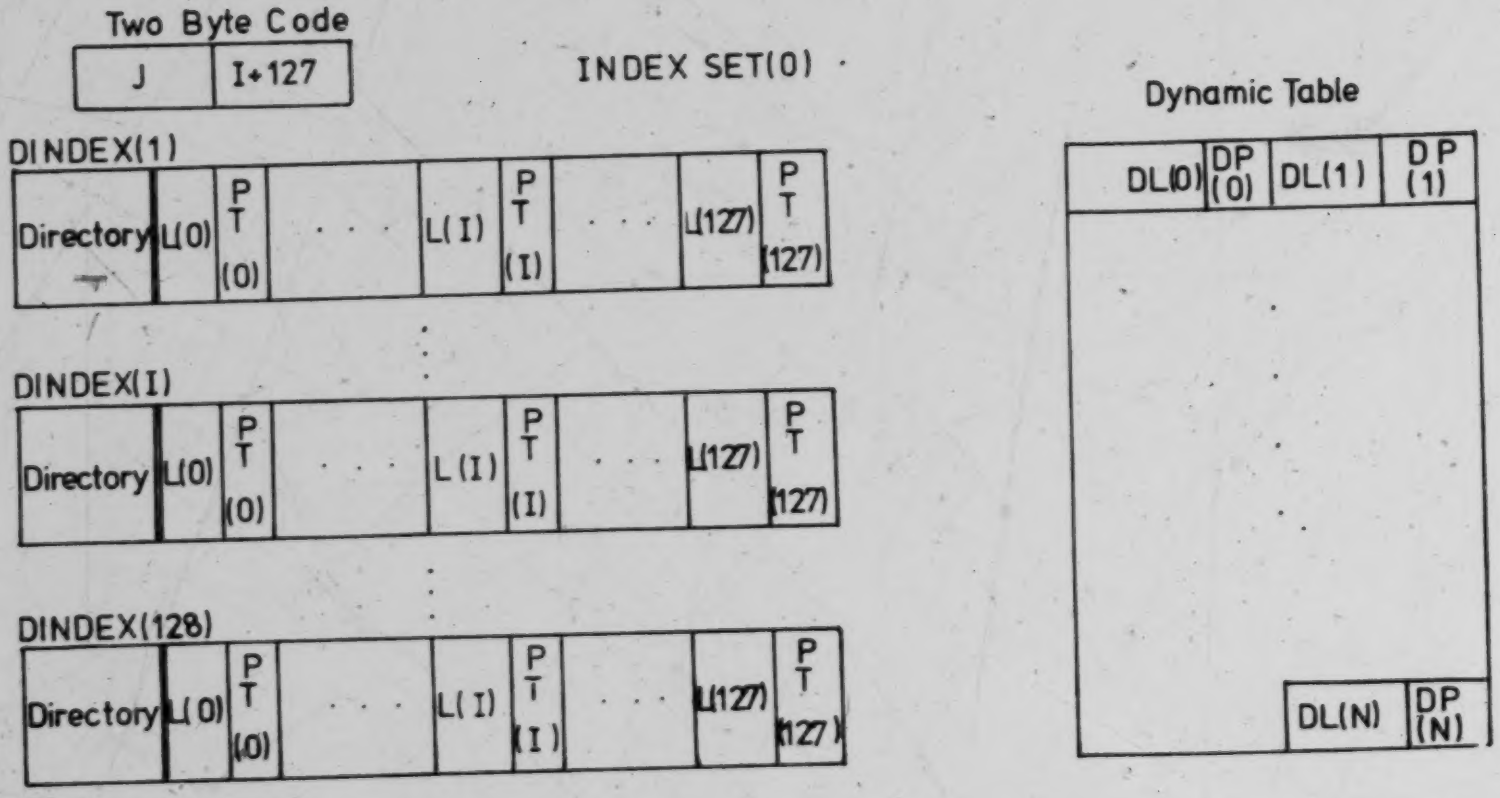

Figure 21

Document Index for Two Byte Codes

steps taken are the same as those performed when PT(i) was examined.

At intervals, depending on the frequency of updates to the document lists, the dynamic lists are merged with the corresponding index lists to obtain new index lists with pointers reset to zero, and empty dynamic tables. This merge is done to keep the dynamic tables down to a manageable size and to limit the amount of chaining and disk reads needed to retrieve a complete document list. To hold search time down, the average number of disk accesses should be kept below 1.3.

3.5.1.3 Deletion of Documents from the Document List

Whenever a record is deleted from the catalogue data base, its corresponding search keys must be removed from the document lists. The searchable words in the record indicate which lists contain the sought after key. The key is then found in each list and set to null. When the merging is performed, all null keys are removed from the lists.

3.6 Evaluation of Dictionary Design

An evaluation of the dictionary's design requires a discussion of both the time taken to perform the various dictionary operations and also the storage space required.

In the present evaluation, it is assumed that tables DIC(0) and HEAD(0) reside in core. It is also supposed that there are 57,800 documents in the catalogue data base. According to Reid and Heaps [27] this corresponds to about 33,000 different words in the title word dictionary. From this, it follows that two sets of tables are required for the three-byte codes.

3.6.1 Expected Decoding Time

The expected decoding time, Td, is defined by:

$$Td = TaA \qquad (3.6.1.1)$$

where Ta is the expected time to read a record from the dictionary file, and A is the expected number of file accesses needed to decode one code.

The length of time to access a disk, 76.6 ms for a 1000-byte record on an IBM 2316 disk pack, renders CPU time insignificant in comparison to disk access time.

The expected number of file accesses required per code during the decoding of title, abstract, and series fields is calculated from Table 1. From this table it follows that the expected number of accesses to decode one code is .45946.

Type of Code	(C)	(B,C)	(A,B,C)
Probability of Code Occurrence	0.49572	0.45544	0.00134
DISK ACCESS			
Header Table	0	0	1
Dictionary Table	0	1	1
TOTAL	0	1	2

Table 1

Decoding Overhead

This number of disk accesses can be reduced if, during the decoding of records, all codes are sorted according to size and value. Then any table will be accessed at most once.

3.6.2 Expected Encoding Time

The number of disk accesses needed to find the table, and the number of probes needed to find the word in that table, are the main factors that govern the time to encode a word. Table 2 lists the disk accesses and probes needed to encode a word according to which table contains the word. From Table 2 it follows that .46020 disk accesses are used to encode a word. Likewise, 17.39 probes will find the word to be encoded.

If the words are alphabetized before being encoded then the number of disk accesses may be substantially reduced. Encoding of words that are in the same dictionary then requires only the number of disk accesses needed to encode one word. Also all words that are to be encoded into a three-byte code, and are in the same set, need only one read of that set's header table.

POSSIBLE WORD LOCATIONS	DIC(0)	SET(0)	SET(1)	SET(2)	NOT IN DICTIONARY
PROBABILITY OF WORD LOCATION WORD LOCATION	0.49572	0.45544	0900096	0.00038	0.04740
DISK ACCESSES					
Header Table	0	0	1	1	2
Dictionary Table	0	1	1	1	3
TOTAL	0	1	3	5	5
SEARCH PROBES					
Header Table	6	6	6	6	24
Dictionary Table	6	6	6	1	22
TOTAL	12	25[1]	38[1]	45	46

[1] Totals take into consideration that seven probes were needed to confirm that the word does not appear in the previously searched dictionary.

Table 2
Encoding Overhead

3.6.3 Expected Updating Time

It takes five disk accesses and forty-six binary search probes to determine that a word does not appear in the dictionary. A new word may then be added to the last table read into core. The position of the new word in that table is indicated by the last search probe. Hence, once it is established that the word does not exist in the dictionary, one disk access is needed to rewrite the amended table onto disk.

3.6.4 Expected Search Time

The dictionary's contribution to the expected search time of full words is through the time taken to establish a pointer to the search word's document list. Since this pointer is the compressed code for the search word, the time taken is the same as that required to encode the word. An analysis of encoding time is given in Section 3.6.2. To this time, the time needed for an additional 1.3 disk accesses are added to account for the retrieval of the document list.

A search specified in terms of right truncation involves more than one document list pointer. Each set of tables must be examined to find all the appropriate document list pointers. From Table 2 it follows that a total of five disk accesses and forty-six binary search probes are needed

to find all the document pointers for the right truncated search.

3.6.5 Estimated Size of the Title Word Dictionary

The amount of disk storage space required for the title word dictionary of 33,000 different title words can be estimated as follows:

i) Reid and Heaps [27] found that the mean length of different title words is 7.6 characters. The term section of the dictionary tables can be expected to require $7.6 \times 3.3 \times 10^4 = 25.08 \times 10^4$ bytes of storage.

ii) The address section of each table requires 260 bytes. There are 385 tables, hence the total storage required for the addresses is $260 \times 385 = 9.95 \times 10^4$ bytes.

iii) The convert code pointers for the three byte code tables require an additional 32,768 bytes of storage.

iv) Each header requires 8 bytes for each of the 128 tables to which it points. Four bytes are used to store the first four characters of the first word of each table. The remaining four bytes are used to store the read pointer for each table. This requires an additional $1024 \times 3 = 3072$ bytes of storage.

v) It follows that the expected size of the complete title word dictionary file is $25,08 \times 10^4 + 9.93 \times 10^4 + 3.28 \times 10^4 + .30 \times 10^4 = 38.59 \times 10^4$ bytes.

3.6.6 Conclusion of Evaluation

The previously mentioned figures regarding speed of access and storage space have little meaning by themselves. It is necessary to compare them to those of other systems which use the same compression technique.

The only other systems that use the selected coding scheme are the system designed by Thiel and Heaps [13] and the system described by Dimsdale [1] and Dimsdale and Heaps [31].

Tables 3 and 4 compare these two systems with the proposed system. These tables clearly indicate the superior design of the proposed dictionary for on-line retrieval systems.

The one area where the proposed system does not excel is that concerned with searching on full terms. However the hashing technique of Dimsdale [1] would lose all it gains in 675 full term searches on one truncated search.

It should be noted that the comparison may not be fair in that the Thiel and Heaps system was originally

designed as a sequential, magnetic tape, batch oriented system.

3.7 Compression of Other Fields of the Catalogue Data Base

In addition to the compression of the title, series, and abstract fields, the UDC number, author, and publisher fields should to be stored in a compressed form. Not only does the coding of these fields greatly increase the saving of disk storage, but at the same time it provides a data structure to support the searching on UDC numbers, on authors and on publishers.

The coding scheme used to compress the author fields and publisher field, and in the dictionary design, is the same as used for title words. Hence the preceding discussion about the coding and searching of title words applies to the compression and searching on author words and on publisher words with the exception of the amount of disk space saved. This is due to the difference in average length and frequency of author words and publisher words. The saving in disk storage can be calculated in the following manner.

	PROPOSED	THIEL & HEAPS	DIMSDALE
SPACE REQUIREMENT (in bytes)			
Dictionary Size to Support a Compressed Data Base	38.49×10^4	26.10×10^4	26.10×10^4
Additional Space to Support Searching	0.0	31.68×10^4	35.43×10^4
TOTALS	38.49×10^4	57.78×10^4	61.53×10^4

Table 3

Space Requirement Comparison

	PROPOSED	THIEL & HEAPS	DIMSDALE
DECODING			
disk access	0.45946	0.45946	0.45946
search probes	1.0	1.0	1.0
ENCODING			
disk access	0.46020	2.28372	2.28372
search probes	17.39	293.58	293.58
UPDATING			
disk access	5	14	14
search probes	46	1,284	1284
SEARCHING			
Full Term			
disk access	1.76020	165	1.01
search probes	17.39	33,000	1.01
Right Truncation			
disk access	8.9	165	515
search probes	48	33,000	33,000

Table 4

Time Comparison

From the 57,846 titles of the MARC tapes issued over the period March 1969 to May 1970, there were 53,387 titles which contained a publisher field; 11,527 of them were different [27]. From this data it follows that.

$P(1) = 0.4091$

$P(2) = 0.5909$

$P(3) = 0.0$

$\bar{c} = 1.5899$

$\bar{w} = 22.6$

$Rs = 14.22.$

This value of Rs implies that the compressed code occupies only 7.03% as much space as is occupied by the publisher field stored in non-compressed form. In terms of disk storage this represents a saving of 1.113×10^6 bytes for 57.8×10^3 titles. In terms of costs at the U of A Computing Services, this is a saving of $274 in disk storage per month.

With regard to the saving acquired in the compression of the author field, the following figures are attained from Dimsdale [1];

$\bar{w} = 5.0$

$\bar{c} = 3.0$

$Rs = 2.0.$

Even though these figures imply only a 50% saving in disk storage used by the author fields, it should be noted that a author dictionary is needed to support searching on author

words. Thus, in fact, the smaller storage saving in compression of the author fields is not lost in overhead.

If the UDC number dictionary is designed such that the UDC schedule's subject words are linked to their corresponding UDC numbers then this structure will also support searching on subject words without having to re-classify each document as to its subject. This UDC number subject word dictionary contributes greatly to the power and versatility of the system. A detailed description of this structure is given in Chapter IV.

CHAPTER IV

THE UDC NUMBER DICTIONARY

4.0 Introduction

In a UDC based library the indication of a document's subject matter is expressed through the UDC number. By use of the UDC schedules the UDC numbers of any document can be translated into English subject descriptors called subject words.

In the present design a dictionary is used by the programs that search for, and compress, the UDC numbers. If this dictionary is linked to a subject word file it greatly facilitates the problem of searching on subject words. It allows a search for a subject word to consist of a translation of the subject word to its corresponding UDC number. The problem is then reduced to that of searching on UDC numbers.

To allow this type of subject searching, the UDC Number Dictionary may be divided into two parts; the Subject Word File, and the UDC Number File. Combining these two parts provides the details of the UDC schedules in use by the library.

4.1 Coding Scheme for UDC Numbers

The faceted structure of the UDC notation allows a UDC library to use a much smaller classifying schedule then

needed by a comparable LC based library. In fact the Boreal Institute Library use s less then 4,000 different UDC numbers.

Because of the relatively low number of different UDC numbers, they can all be compressed into a code of two bytes. This code will be denoted by (A,B) where $0 \leq A \leq 255$, and $0 \leq B \leq 255$. It should be noted that since all codes are of the same length, a flag byte is not needed to separate the codes.

4.2 Design of the UDC Number File

The UDC numbers presently in use at the Boreal Institute library may be ranked in a list according to the UDC filing order. The list may then be split into lists of length 256 which are successively placed in tables UDC(0),UDC(1),...,UDC(N) until fewer then 256 UDC numbers remain. The lists of UDC numbers, U(n) $0 \leq n \leq 255$, start at byte 1026 of the table.

The two-byte address, A(n) $0 \leq n \leq 256$, for each UDC number is inserted into the table starting at the first byte of the table. These addresses are the same as those described in Section 3.1.1 except that there are 257 relative to 1026.

The 256 two-byte fields, SP(n), $0 \leq n \leq 255$, that follow the addresses will be used to point to each UDC

number's subject list. The use of this part of the table will be explained in Section 4.6. The design of a table, UDC(i) from this set is shown in Figure 22.

Two dynamic tables, UDC(N+1) and UDC(N+2) are created to hold the remaining UDC numbers and to facilitate updating. Table UDC(N+1) is initially empty while table UDC(N+2) initially contains the remaining UDC numbers. The design of these tables is illustrated in Figure 23 where the first 512 bytes are used as addresses, A(i), the next 256 bytes are used as subject list pointers, SP(i), the next 256 bytes are used as convert codes, CC(i), and the remainder of the table contains the UDC numbers.

The convert codes in these two tables perform the same function as those described in Section 3.1.3. That is, they are used to indicate the last byte of each code.

When either of these two tables becomes full, then the pair of tables is considered to be complete and another set of two dynamic tables is created and used for further updating.

To allow direct access to the tables, a header table UDC-HEAD is created in the form shown in Figure 24. The C(i), $0 \leq i \leq N$, are the corresponding first four characters of the first UDC number of each table UDC(i). C(N+1) is set to four blanks, and C(N+2) is set according to a function which takes into account the split of the occurrences of the

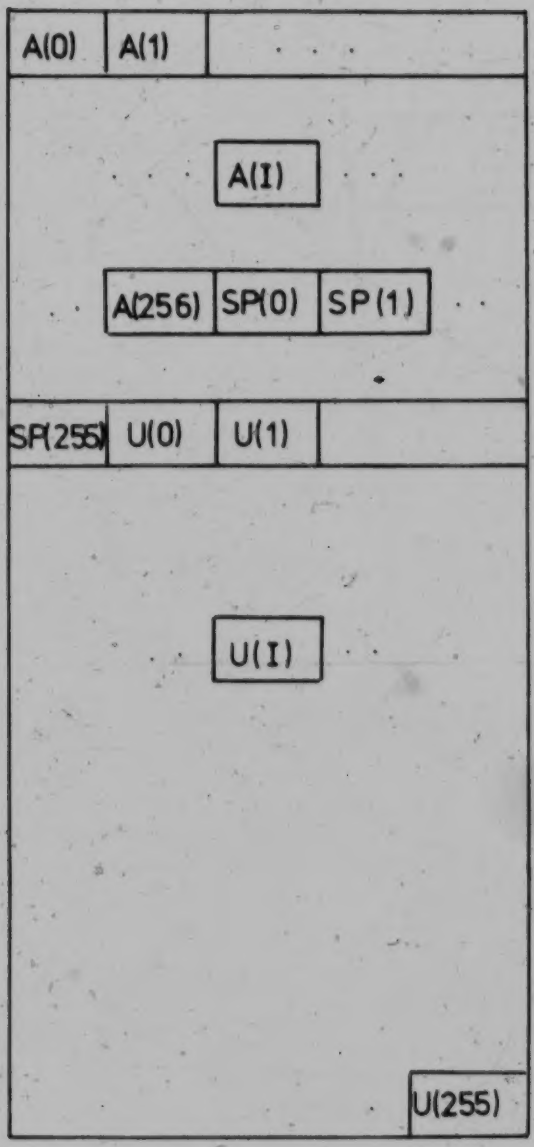

Figure 22

Stable UDC Number Table

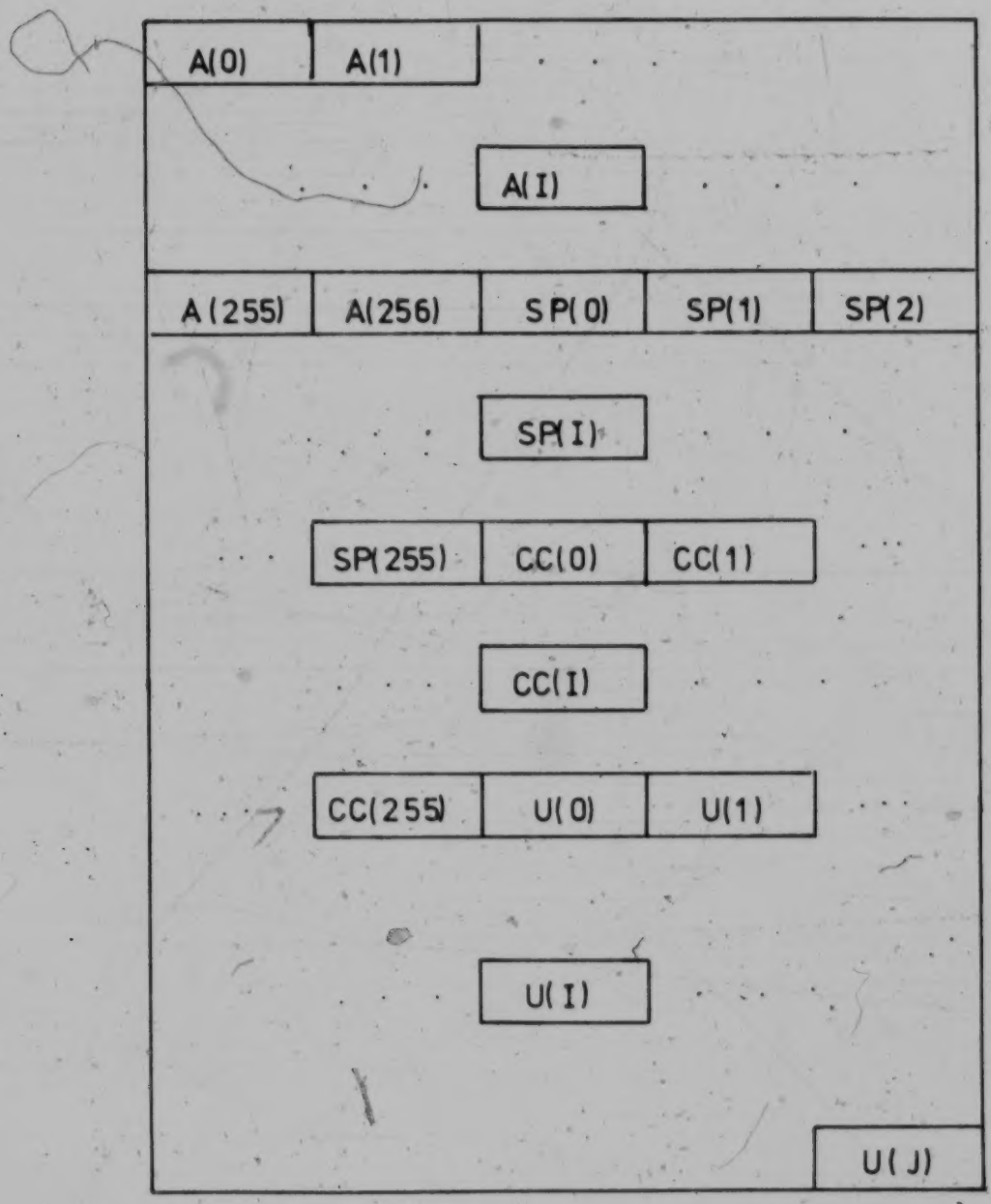

Figure 23
Dynamic UDC Number Table

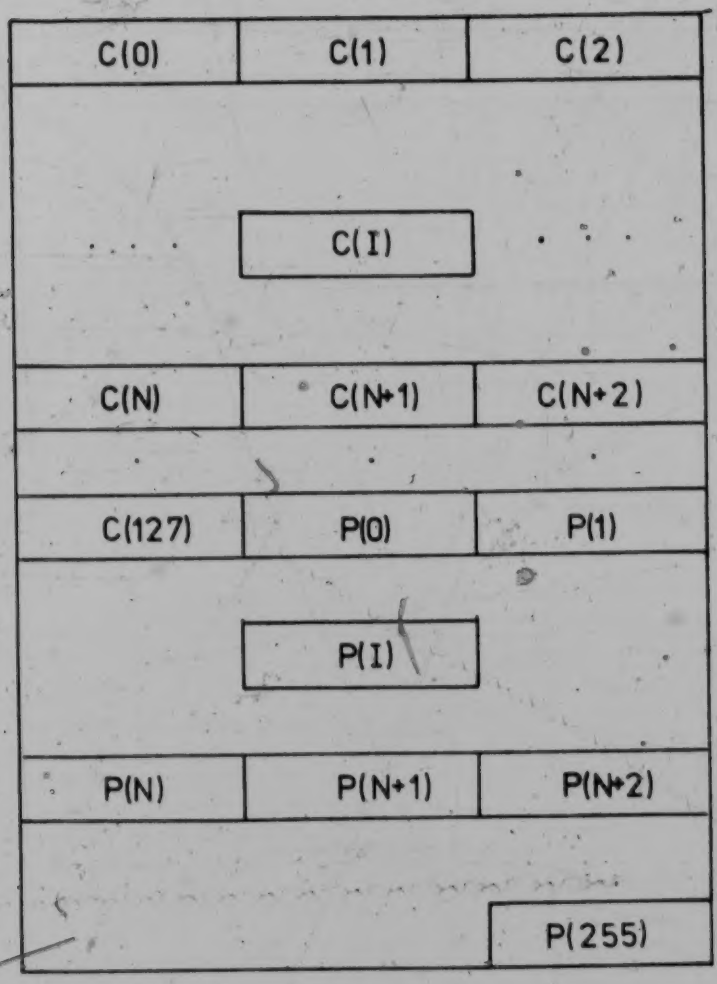

Figure 24
UDC-HEAD

UDC numbers and the number of UDC numbers already in table UDC(N+2). The pointers, P(i) contain the relative disk locations of the respective tables.

Figure 25 shows the complete set of tables UDC(i), $0 \leq i \leq N+2$, and its corresponding header table UDC-HEAD.

4.3 Operations Performed on the UDC Number File

The various dictionary operations for UDC numbers are performed basically in the same manner as the similar operations for title words.

4.3.1 Decoding UDC Numbers

To decode a UDC code, the first byte is used to indicate the table that contains the UDC number. By use of the read pointer in UDC-HEAD, that table is then read into core. The second byte of the code indicates the position of the UDC number in the table. Using the address of the table, UDC(i), the appropriate UDC number may be found.

If the first byte of the code indicates a table with a value greater than N, then the corresponding table is a dynamic one. Search of a dynamic table makes use of the convert codes in the same manner as was described in Section 3.4 to locate the proper UDC number.

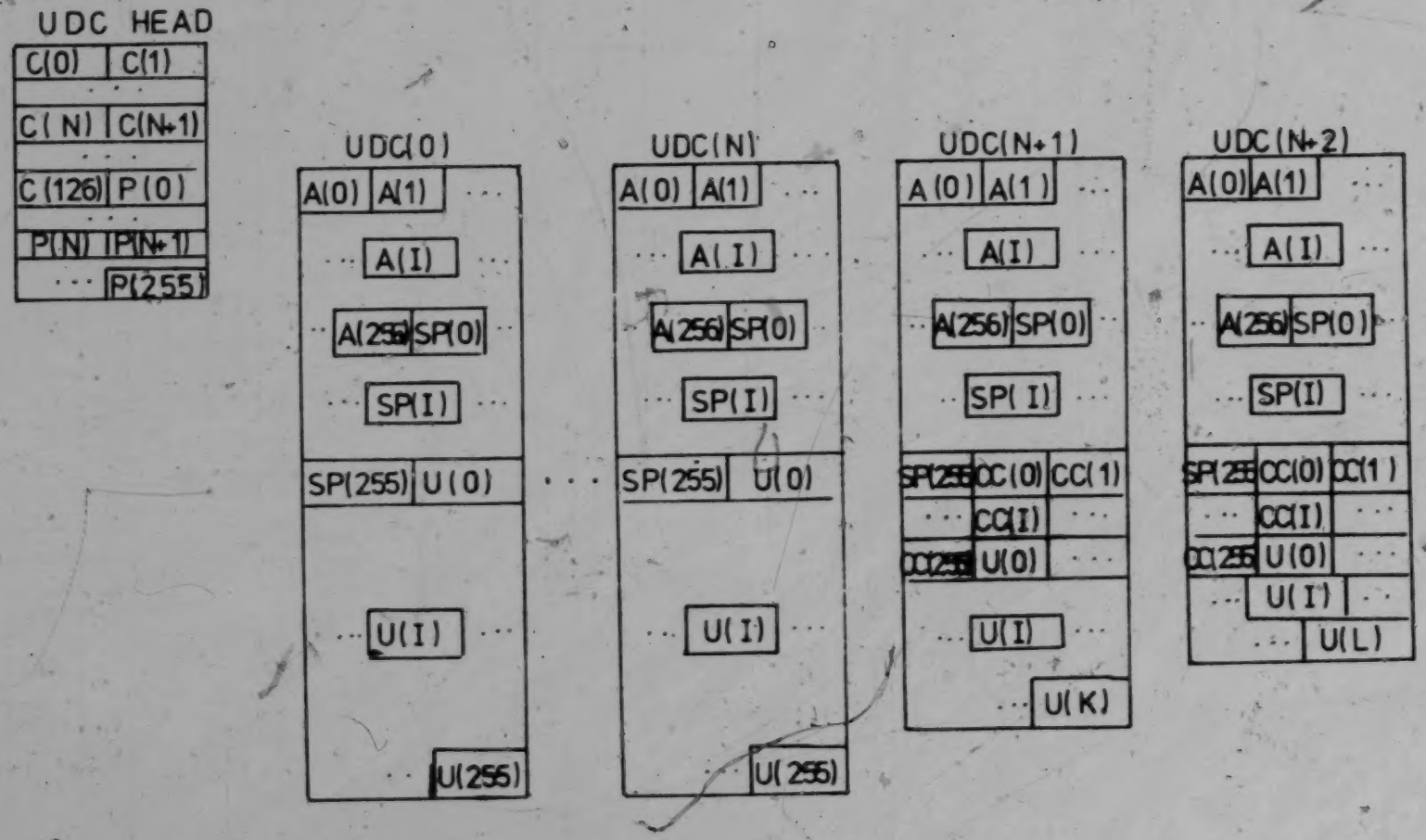

Figure 25
UDC Number File

4.3.2 Encoding UDC Numbers

To encode a UDC number, a binary search is performed on the set C(i), $0 \leq i \leq N$, of UDC-HEAD in order to determine which UDC table may contain the UDC number. This table is read into core and a binary search is performed on it to locate the word. If the number is found, then the first byte of the code is the value of the table that contains the number. The second byte of the code is the position of the word in the table.

If the UDC number is not found in tables UDC(i), $0 \leq i \leq N$, then the UDC number is compared to C(N+2) of UDC-HEAD. If the UDC number is less; then table UDC(N+1) is read into core. Otherwise table UDC(N+2) is read into core.

A binary search is performed on the table that was read into core. If the word is found it is then assigned a code as before, except that the second byte is assigned by the convert code. If the number is not found, and there is another set of two dynamic tables, then these are searched for the number in the same manner. When there are no more sets of dynamic tables to examine, then it is known that the particular UDC number has not been included in the UDC Number File.

4.3.3 Updating the UDC Number File

When it is known that a UDC number does not appear

in the UDC Number File, then the number to be added is compared to the header character for the last table of the last set of dynamic UDC tables. If the number to be added is less; then the first table of the set is read into core. Otherwise the second table of the set is read into core.

After the table to which the new number to be added is put into core, the process of inserting the new number is the same as described in Section 3.2.1 for the insertion of new title words in the Title Word Dictionary.

4.3.4 Searching on UDC Numbers

The method of searching on UDC numbers is identical to that for searching on title words. The UDC number to be searched on is first encoded. The code points to the list in the UDC Document Index that contains all documents that are referred to by that UDC number.

The UDC Document Index is designed similarly to the Title Word Document Index. The only difference is that each section of the index contains 256, rather then 128, lists.

It is possible to search on right truncated UDC numbers because the UDC numbers in the UDC Number File are arranged in UDC filing order. The process is the same as that used to search on right truncated title words, all sets of tables being examined for codes of all UDC numbers that start with the desired root. The UDC Document List for each

code is retrieved from the UDC Document Index. These document lists are "OR"ed to produce a resultant document list that contains a list of all documents that satisfy the right truncated search.

4.4 Design of the Subject Word File

All subject words presently in use at the Boreal Institute Library may be arranged in alphabetical order. These subject words are then inserted in order into tables SUB(0),SUB(1),...,SUB(N). The design of table SUB(i), given in Figure 26, is of similar form to table UDC(i).

The codes to the subject word's corresponding UDC number are contained in the 512 bytes, starting at position 514. Thus the UDC pointer, UP(i), contains the code of the UDC number to which the subject word S(i) refers.

However the correspondence between subject words and UDC numbers is not always one-to-one, thus the two-byte code, UP(i), must allow location of more then one UDC number. The tables, UDC(j), $128 \leq j \leq 255$, are created as needed to contain the locations of the additional UDC number locations.

The 512 byte tables UDC(j), $128 \leq j \leq 255$, are divided into sections of four slots. Each slot is two bytes long. The four slots will contain up to four UDC number codes. If a list of more than four locations is required,

Figure 26
Subject Word File

then the fourth entry is a pointer amend a UDC number requires only a to another set of up to four UDC number codes.

If there is only one UDC number corresponding to the given subject word, then UP(i) will contain the code of that UDC number. If there is more then one UDC number corresponding to the given subject word, then UP(i) will give the location in table UDC(j), $128 \leq j \leq 255$, of the list of corresponding UDC number codes.

The distinction as to whether UP(i) contains a UDC number code or a pointer to a list of UDC number codes is made by the value of the first byte of UP(i). If this first byte is less then 128, then UP(i) contains a UDC number code, or else it points to a list of UDC number codes in UDC(j).

Figure 21 illustrates the structure of the Subject Word File and tables UDC(128),UDC(130),...,UDC(255). For these tables, direct access is obtained by means of the read pointer P(j), $128 \leq j \leq 255$ of UDC-HEAD.

4.5 Operations Performed on the Subject Number File

The operations of decoding and encoding are performed in the same manner as outlined in Section 4.3 for the UDC number file.

4.5.1 Updating the Subject Word File

The only way in which updating the Subject Word File differs from updating the UDC Number File is that when a new subject word is inserted in position i, all UDC pointers UP(j), $i \leq j \leq 256$ are shifted down one position. The UDC pointer for the new subject word is inserted in UP(i).

4.5.2 Searching on Subject Words

To search the Catalogue Data Base on a subject word, the subject word is first found in the Subject Word File. This procedure is the same as that of encoding a subject word.

Once the subject word is found, its corresponding UDC pointer, UP(i), is located. All UDC codes indicated by UP(i) are then retrieved. These codes indicate the required lists to be retrieved from UDC Document Index. Such lists are fetched and "OR"ed together to obtain a resultant list of all documents that are referred to by that subject word.

Since the subject words are arranged in alphabetical order, it it very easy to search on right truncated subject words. This search is performed in the same style as the search on right truncated UDC numbers or title words.

4.6 Translation of UDC Numbers to Subject Words

Whenever a new document is catalogued, a check must be made to ensure that the assigned UDC numbers accurately describe the subject matter of the document. There is therefore a need to translate a UDC number to its corresponding subject word. To achieve this translation there must be a link between each UDC number and its corresponding subject words.

The link is accomplished through the use of the SP(i)'s, $0 \leq i \leq 255$, of each table UDC(j), $0 \leq j \leq N$, in the UDC Number File. The SP(i)'s of the UDC Number File are used exactly in the same manner as the UP(i)'s of the Subject Word File. Each SP(i), $0 \leq i \leq 255$, contains either the code of the UDC number's corresponding subject word, or a pointer to a list of subject codes in the 512 byte table SUB(j), $128 \leq j \leq 255$. The complete UDC Number File with the linkage to the subject word file is shown in Figure 27.

With this pointer between a UDC number and its corresponding subject words, the desired translation is straight forward. The UDC number to be translated is found in the UDC Number File. The corresponding subject word codes for that UDC number are retrieved. This list of subject word codes is then decoded to the appropriate subject words.

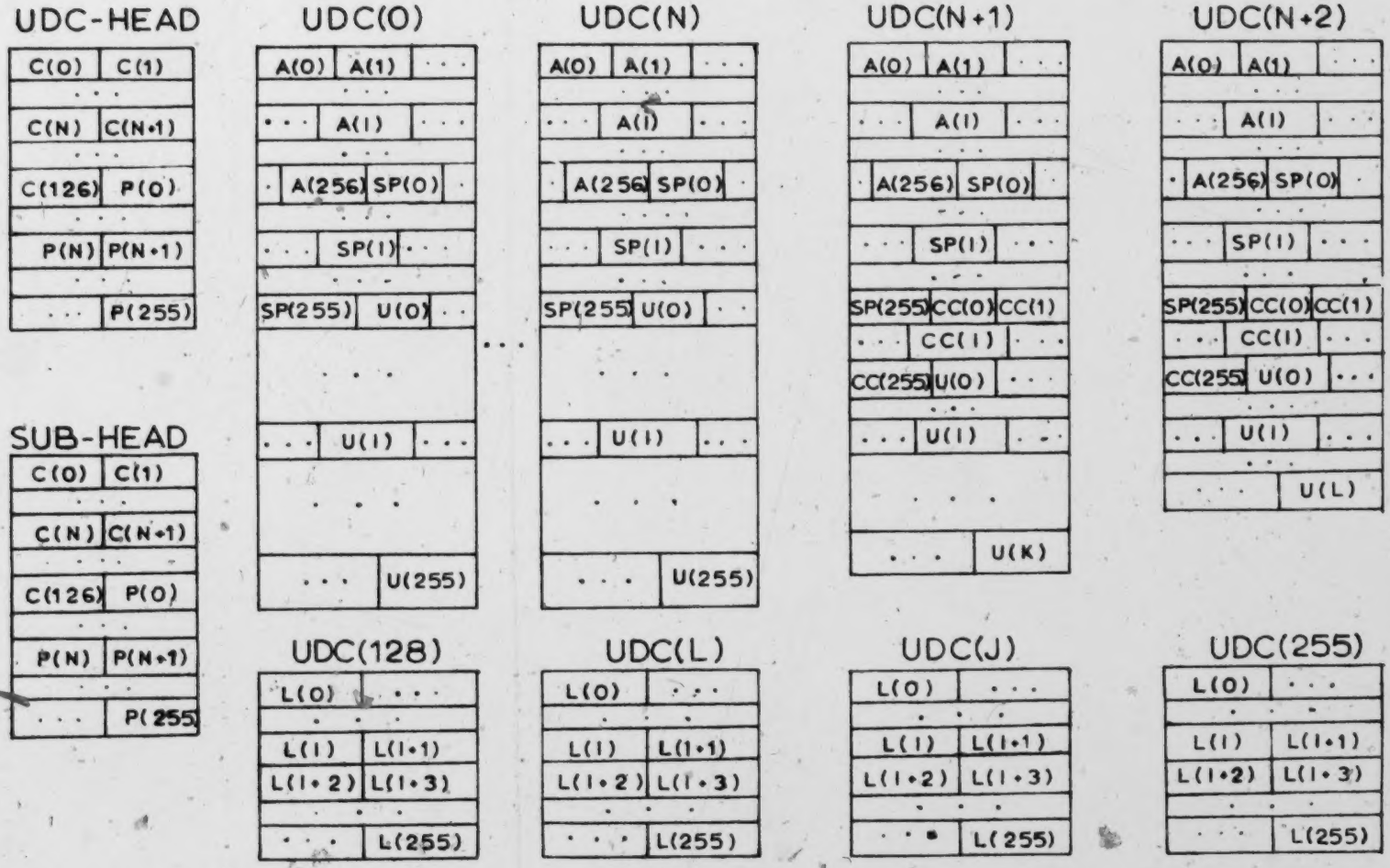

Figure 27
UDC Number File

It should be noted that when a new UDC number is added to a dynamic UDC table at position i, then all SP(j), $i \leq j \leq 255$, are shifted down one position. The subject pointer for the new UDC number is inserted at SP(i).

4.7 Translation of Subject Words to UDC Numbers

In the process of classifying a document, its subject content is translated to UDC numbers. To aid in this phase of classifying, the subject words must be translated into UDC numbers.

The procedure used to execute this translation is the same as that used to search on subject words. However, after the UDC number codes are found, instead of retrieving the corresponding document list, the codes are decoded to obtain the subject word's corresponding UDC number.

4.8 Amendments to UDC Numbers

The nature of the UDC schedules allows the librarian to construct a UDC number to express a new subject. When this is done, it often happens that a different UDC number is officially assigned to that subject or conversely a different subject is officially assigned to that UDC number. After this change in UDC schedules, all books referenced to by that UDC number must be re-classified. Likewise, the

corresponding records in the Catalogue Data Base must be changed.

If the UDC numbers were not coded the task of changing UDC numbers in the Catalogue Data Base would be very costly. The problem lies in the fact that UDC numbers are not of a fixed length. Thus when a new UDC number is longer then the number it replaces, the complete Catalogue Data Base would have to be reconstructed. But since the UDC numbers are all coded in two bytes, to amend a UDC number is to change two bytes in the Catalogue Data Base.

To facilitate searching on the amended UDC schedules, changes must be made to the UDC Document Index. These changes consist of deleting from the old UDC number's document list the keys for all documents which were re-classified. These keys for the re-classified documents are then inserted into the new UDC number's document list.

The search on subject words and the translations between UDC numbers and subject words must be kept in agreement with the current UDC schedules. To obtain this consistency, changes are made in the subject pointers and the UDC pointers. The code of the old UDC number in the subject word's corresponding UDC pointer list is replaced by the code of the new UDC number. The code of the subject is deleted from the old UDC number's subject pointer list, and is added to the new UDC number's subject pointer list.

4.9 Thesaurus for a UDC Based Library

The facet features of UDC, the arrangement of UDC numbers in UDC filing order within the UDC Number File, the ability to search on units of the classification, and the prevailing hierarchical correspondence between the UDC notation and the subjects they represent, make it very easy to create a thesaurus directly from the UDC Dictionary. All that is required is a small program to manipulate the UDC filing order and the relationships between subject words and UDC numbers. Some relationships between subjects and UDC numbers have been investigated by Mercier [5].

For every subject word in the subject word file, the thesaurus should provide the subject words that satisfy the relationships of synonyms, broader terms, related terms, and narrower terms.

i) Synonyms are defined as all subject words referred to by the same UDC number that refers to the given subject word. To obtain the synonyms of a subject word, that subject word's UDC pointer list is retrieved. The subject pointer lists of all the UDC codes in the UDC pointer list are ORed. Except for the code of the given subject word, all subject codes of the resulting list are decoded to provide the appropriate synonyms.

ii) The set of broader terms annotated with a given term is defined to include all subject words which are referred to by a right trucated UDC number of the given subject word. To obtain all broader terms the given subject word is translated to its corresponding UDC numbers. The last digit of each of the corresponding UDC numbers is then deleted. The new UDC numbers are then translated to subject words. These subject words are the broader terms of the given subject word.

iii) Related terms are subject words referenced by UDC numbers in which only the last digit differs from the UDC number of the given subject word. The related terms can be obtained by first retrieving the UDC numbers of the given subject words. A right truncated search is made for all UDC numbers of the same length in which only the last digit differs from the UDC number of the given subject word. These UDC numbers are translated to give the related terms for the given subject word.

iv) Narrower terms are defined as subject words referenced by UDC numbers that are the same as UDC numbers of the given subject word except that the referring UDC numbers are one digit longer. To find the narrower terms of a given subject word, the subject word is translated to its corresponding UDC numbers. These UDC numbers are then used in a right

truncated search for all UDC numbers that are one digit longer. The longer UDC numbers are translated to provide the narrower terms of the given subject word.

Illustration of the process of forcing a thesaurus from the UDC schedules is provided by Figure 28 which shows a section of the UDC schedules and the resulting thesaurus.

It is worth noting that it is impossible to provide the above type of thesaurus on an automated UDC based library which does not keep the UDC numbers in UDC filing order or which fails to support either of the operations of a fast right truncated search, or the translation between UDC numbers and subject words. For such UDC based libraries, and all non UDC based libraries, a sizable addition to the data base is needed to support a thesaurus. If a thesaurus is needed then the programs by Alber [6,31] may be used. They provide for creation, update, and use of a more comprehensive thesaurus which does not rely on the UDC classification system.

4.10 Evalution of the UDC Number Coding Scheme

In addition to providing an easy method of changing UDC numbers in the Catalogue Data Base as described in Section 4.8, and providing a efficient means of linking UDC numbers to subject words, as described in Sections 4.6 and

SECTION OF UDC SCHEDULES

536.7 Thermodynamics
Heat energy

.71 Change of state. Equations
.711 Adiabatic
.712 Isothermal
.713 At constant pressure
.714 At constant volume
.715 Under other conditions
.717 Cycles. Diagrams
.72 First law of thermodynamics
Equivalence of forms
Law of conservation of energy
.721 Mechanical equivalent of heat
.722 Energy forms
Enthalpy
.73 Second law of thermodunamics
.73 Thermodynamic temperature

RESULTANT THESAURUS

SUBJECT
Equivalence of forms

SYNONYMS
First law of thermodynamics
Law of conservation of energy

BROADER TERMS
Thermodynamics
Heat energy

RELATED TERMS
Change of state. Equations
Second law of thermodynamics
Thermodynamic temperature

NARROWER TERMS
Mechanical equivalent of heat
Energy forms
Enthalpy

Figure 28

Thesaurus and its Corresponding UDC Schedules

4.7, the coding of UDC numbers provides a substantial saving in disk storage space.

4.10.1 Disk Storage Saving Attributed to a Compressed UDC Number Field

With the average length of a UDC number being 11.6 characters and with an expected compressed code length of two bytes, the data compression ratio expressed in equation (2.3) is found to equal 5.8. This implies that the compressed data fields occupy only 17.2% as much space as is occupied by the same data stored in a non-compressed form.

This saving can be expressed in terms of disk storage saved for 50,000 titles as follows:

i) With an average of eight UDC numbers appearing in each document record, the total space needed to store the uncoded UDC numbers is:
11.6 x 8 x 50,000 = 4,640,000 bytes.

ii) The space needed to store the compressed UDC numbers is 2 x 8 x 50,000 = 800,000 bytes.

iii) It follows that the compressed code represents a saving of 3,840,000 bytes of disk storage for 50,000 documents. Using the University of Alberta Computing Services's charging rate, as given in Section 1.1.1, the saving in disk storage represents a saving of 3,840,000 bytes x $1.00/ 4096 byte month = $937.50 per month.

To obtain a net estimate of the saving in disk storage produced by use of a compressed UDC number field, the storage saving must be weighed against the overhead in terms of CPU time and disk storage required to support the compressed UDC number field. Section 3.6.1 showed that the CPU time required to decode items was sufficiently small as to not effectively increase the operating cost of the system. A comparable amount of disk storage to that used by the UDC Dictionary would be required by a non compressed code retrieval system to hold a file structure to allow searching on UDC numbers and subject words. Thus it can be concluded that the net saving produced by a compressed UDC number field for 50,000 documents is over three million bytes of disk storage.

4.11 Evaluation of the Design of the UDC Number Dictionary

The design of the UDC Number Dictionary provides the system with an extremely powerful device. It has been shown in this chapter that the UDC Number Dictionary supports a wide range of on-line activities. The operations are:

i) The full text and right truncation searching on UDC numbers.

ii) The full text and right truncation searching on subject words.

iii) The creation and support of a compressed UDC number field.

iv) The translation of UDC numbers to subject words.

v) The translation of subject words to UDC numbers.

vi) The maintaining of an up to date UDC schedule in on-line mode.

vii) The forcing of a thesaurus from the UDC schedules to provide an aid in the searching of subject words of the the on-line catalogue.

Not only does the UDC dictionary perform all these operations but at the same time it is economical in terms of storage requirements of the dictionary and in terms of CPU time needed to provide these operations. Since there does not exist another on-line library system which supports all these operations, it is very difficult to compare the economy of the UDC dictionary.

To illustrate the economy of storage requirements of the dictionary, it may be remarked that Alber's [6,31] design of a very efficient method of providing only a thesaurus, or Dimsdale's [1] hash address technique to provide only full word searching on UDC numbers and subject words, requires more disk space then used by the UDC Dictionary.

To illustrate the economy of CPU time, Table 5 shows the number of disk accesses and search probes needed

	DISK ACCESSES	SEARCH PROBES
DECODING	1	1
ENCODING	1	7
UPDATING		
UDC Number file	3	14
Subject Word File	3	14
TRANSLATION		
Subject Word to UDC Number	2	8
UDC Number to Subject Word	4	9
SEARCHING ON UDC NUMBER		
Full Term	2.3	7
Right Truncation	3.3	14
SEARCHING ON SUBJECT WORD		
Full Term	2.3	8
Right Truncation	3.3	14
CREATE A THESAURUS[1]	12	20

[1] To produce the example in Figure 23

Table 5

UDC Dictionary Overhead

to perform the various operations. Taking into account that the time taken for one disk access of a 1,000 byte record is 71.6 ms and the average time taken for a search probe is 15 micro sec., it becomes quite apparent that all the UDC dictionary operations will have very short response time.

From the above discussion, it becomes apparent that even with the fact that the design of the UDC Number Dictionary provides more on-line activities for a UDC based library than any other file structure, it does so with less use of disk storage and CPU time then used by other systems which only provide a few of the same activities.

CHAPTER V

PROGRAMMING CONSIDERATIONS

5.0 Introduction

The file structure developed for this study was implemented in 360 Assembler language on the University of Alberta Computer Center's IBM 360/67. This computer is controlled by a large batch-terminal multi-processing operating system called the Michigan Terminal System (MTS).

The purpose of the implementation was to demonstrate that the data structure developed in this study can be freely implemented to provide an on-line automated library system. The result provides a meaningful test of the basic design's flexiblity and power since the implementation completes an on-line automated UDC library system for the Library of the Boreal Institute for Northern Studies.

The design of the system is modular as there is a separate routine to control each of the commands described in Appendix I. There is also a library of routines used by the command routine to perform the following functions.

i) Indicate the next table that should be looked at to find a given word.

ii) Read into core the desired dictionary table.

iii) Search an in-core dictionary table in order to locate a specific word or a set of words that satisfy a given right truncation.

iv) Decode a record from the compressed catalogue data base.

v) Encode a new holding into the compressed catalogue record format.

vi) For a given compressed code, find its corresponding inverted index list.

Figure 29 illustrates the overall structure of the system.

The implementation was programmed in 360 Assembler language. Assembler language was chosen because of the great ease of byte mainpulation not available through use of most higher level programming languages. Assembler language was also used because a large on-line retrieval system needs a high efficiency in use of computer time in order to obtain minimal response delay.

The IBM 360/67 can serve a large number of users by concurrently offering each one a wide variety of services. The task of keeping track of all the programs in the machine, while devoting some attention to each of them every second or two, is handled by the MTS operating system. In addition to execution control, MTS handles command interpretation, accounting maintenance, and file management.

The implementation of the file structure into an on-line retrieval system directed this study to the investigation of two areas which must be considered in the

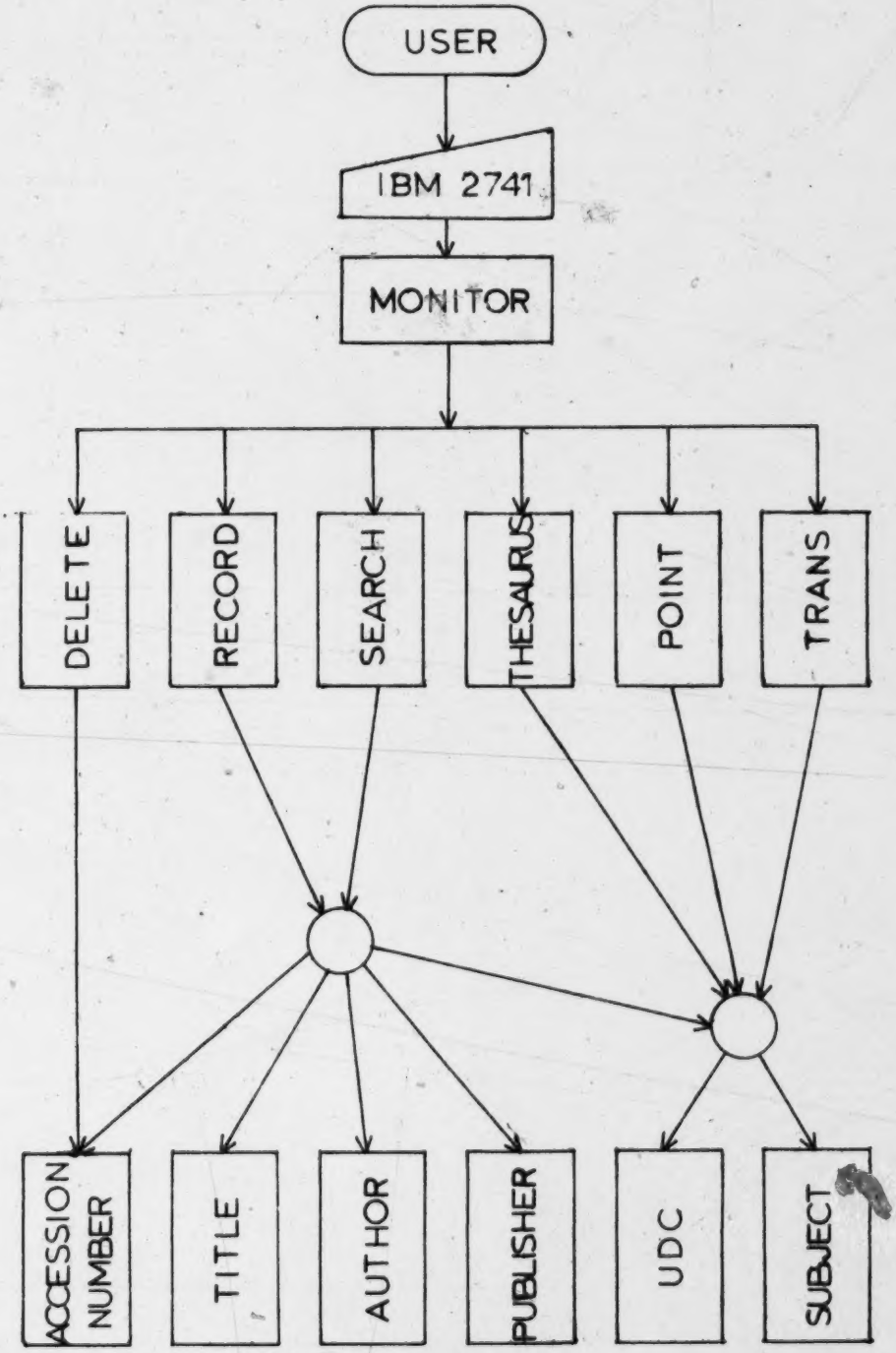

Figure 29

General Structure of the On-Line System

total design of an on-line retrieval system. They involve obtaining of the best possible disk access time and the protection of the data base during system failure. The remainder of the present chapter will present a design of these two segments of an on-line system. It may be remarked that lack of access to a hands on computer prevented the implementation of these two features during the present study.

5.1 Improved Disk Access Time

As shown in Section 3.6.5 the greatest overhead for the proposed data structure is the disk access time taken to search for a non-right truncated term. This section will propose a method of storing the dictionaries on disk in order to greatly reduce this overhead.

The greatest component of disk access time is the time required to move the read-write head to the proper cylinder. For example, of the average time of 75.7 ms required to access a 1,000 byte record on a IBM 2314 disk pack, an average of 60 ms are used to position the read-write head.

If the retrieval system is implemented on a computing resource which allows for at least direct control of the disk operations then the average disk access time can be greatly diminished. The object is to store the

dictionary on disk in such a manner that the minimum number of head movements is needed.

For the title word dictionary there are five complete sets of 128, and one set of 128, dynamic tables to be stored on disk. The average size of each table is such as to allow five tables to be stored on one track of a 2314 disk.

With respect to disk access time, the best possible storage method for this size of dictionary would be to store the Nth table of each of the five sets of 128 tables on the same track. This placement is such that by storing the tables track by track, cylinder by cylinder the order of the tables is preserved. With this method of table placement, all words within a certain division of the alphabet would be arranged on the disk such that they can all be accessed with only one head movement.

For the dynamic tables each track is split to provide storage for two consecutive dynamic tables. This allows space for another 40,000 different words. More space will not be needed until the 50,000 document library has expanded to exceed 97,000 documents. Likewise for a dictionary that expands at a slower, or faster, rate the number of different divisions of a track will vary. The searching for 99.526% of all words in the dictionary will require one disk access. A maximum of two disk accesses

will find any word in the dictionary. The search on right truncation will require only two disk accesses. This is a great improvement of the search times given in Section 3.6.4.

5.2 Recovery

All on-line retrieval systems must have a recovery technique. A recovery system will assure that the database is never suspect or lost due to failure of the retrieval system. Such a failure may be due to the presence of unanticipated data, program fault, hardware fault, system overload, partial or total loss of power, or operating system failure. The suggested method of recovery is an adaptation of a system presently being developed at the City of Edmonton's Computer Systems Branch for use with Univac's 1100 Data Management System [33].

To obtain recovery, provisions must be made to save critical or non-reconstructable information in a timely manner. This must be done continually in order to provide sufficient information to allow recovery from any failure of the retrieval system.

5.2.1 Bulk Tape Saves

The most direct approach to providing recovery information is by a periodical bulk tape save of the data

base. The period between saves must be tailored to the amount of alteration incurred through data base usage. The time taken to obtain a bulk save can be greatly improved if the saving mechanism is sensitive to files which cannot be altered. The effect here is that such files, like the stable dictionary tables, need only be saved once.

Bulk save can be characterized as a slow, inexpensive method of providing for recovery. It causes restoration of a large data base to be time consuming. There will also be differences between a saved data base and the operating one. These two factors make bulk save alone inadequate for real time on-line retrieval systems.

5.2.2 Audit Trail Tape

The use of an audit trail tape enhances the bulk save technique and is more suited to an on-line environment. An audit trail is a chronological series of event descriptors. The audit trail tape will contain before looks, after looks, and the change transactions. A before look is a copy of the table before it is modified, an after look is a copy of the table after it is modified, while a change transaction will be the command input that caused the change.

The audit trail tape is not sufficient by itself to protect against a system catastrophe. The reason is that

total data base recovery would involve applying all the after looks in the entire audit trail from the moment of occurrence of the first data base information item. For this reason periodic bulk saves are employed and an audit trail is associated with each save. The recovery consists of loading the most recent save and applying the after looks on the associated audit trail in order to reconstruct the data base.

This method also causes a considerable down time during recovery. This time is directly related to the amount of time needed to reload the data base from the bulk save.

An improvement on the above method is to specify intervals at which break points are written on the audit trail tape. A break point is an indication that the data base was valid at that particular point in time. The resulting method of tape recovery consists of applying, in backward order, the before looks from the time of system failure until a break point is encountered. This undoing of changes from the last break point to the system failure provides a valid data base. The base is made current by applying the after looks in the same manner as described previously. If the system failure was caused by an input programming error, then the data base can be brought to current status by selectively reprocessing the saved transaction input.

This method could still cause down times of upwards of an hour. Such is still unexceptable for an on-line retrieval system.

5.2.3 Disk Scratch Pad

A faster method of recovery is to use disk storage in a scratch pad method to hold before looks of all tables that are modified by any one transaction. The disk storage that contains these copies of tables is released when the command is sucessfully completed. These copies of tables are to be used when a transaction terminates in error. Thus any changes made to the data base by the incomplete transaction can be undone.

When combined with the audit trail and bulk save, the temporary disk save can virtually guarantee that the database of the system is complete no matter how severe the failure. The length of down time required to recover the data base will be kept to a minimum and is proportional to the severity of the system failure.

5.2.4 Continuous System Operation

Another problem that exists during system failure is that of what happens to the user at this time? The solution to this problem depends on the on-line system. For systems such as the one developed in this study, which makes use of

a third party's computer, very little can be done. At best terminals with memories can be used so that change transactions can still be entered and the input saved for processing at a later date.

For on-line retrieval systems which require high system availability then uninterruptable power supply and redundant hardware are necessary.

Redundant hardware is protection against hardware unit failures. The general rule is that central hardware, core and CPU, have one extra unit for each system failure. Peripherals such as disk, printers, and tapes, have one extra unit wherever critical. It may be noted that such extra hardware can be used at will at times when a system failure is not present.

A useful feature of redundant hardware is the ability to have the data base and its dual stored on different devices. Dual storage involves maintaining duplicate copies of the data base on disk. Both copies are actively used. A write operation is done simultaneously to both copies. Read operations are done from the copy which will respond most quickly. If a read fails, the software will attempt retrieval from the other copy and, if possible, will overwrite the information on the first copy in order to make it useable again. Furthermore, and most important, a

mass storage unit failure will not cause any interruption in service if the data has been duplexed on another device.

Redundant hardware tends to be an expensive implementation. The extent of the implementation must be carefully weighed against the unacceptabilty of system down time.

CHAPTER VI

CONCLUSION

The investigation reported in this thesis was concerned with the design and implementation of file structures for use in an on-line retrieval system for a UDC based library.

The primary purpose of the implementation was two fold. The first was to discover any inadequacies in the scope or design of the proposed file structure. The second was to demonstrate the feasibility of implementation of an on-line library system using the file structure as described in Chapters III and IV.

The thesis has shown that not only is the system feasible, but it is more flexible and powerful then are other compressed data base retrieval system such as those described by Dimsdale [1], Heaps and Thiel [13], and Dimsdale and Heaps [31].

It has been shown that the file structure competes well with the other systems. Its power is exemplified in the fact that, with no additional overhead, the dictionaries provide fast right-truncated searching and a thesaurus for UDC based libraries. The total space needed to support all these operations is less then that needed for the other systems.

The flexibility of the data structure is shown by the fact that it can be used by both large and small UDC libraries. It should by noted that the system could be used for a non UDC based library with only loss of the thesaurus and ability to search on subject words. The search on subject words could be regained if the holdings of the library are classified according to subject words or keywords. The flexibility of the design is also shown by the fact that although the file structure was intended mainly for on-line retrieval systems, only a few minor modifications are needed in order to produce a batch retrieval system.

Additional study, not covered in this thesis, would show other uses of the basic file structure. Further study should be done on the following:

i) The modifications necessary to convert the structure to an non eight-bit byte, and four-byte word computer.

ii) The feasibility of forcing a thesaurus out of part of the LC Schedules, based on the work done by Dobay and Heaps [34].

iii) The appropriateness of the basic structure for a retrieval system that makes use of word fragments developed by Schuegraf and Heaps [35]. This type of system would provide an efficient method of supporting right and left truncation searching.

BIBLIOGRAPHY

1. Dimsdale, J.J., On-line Library Automation Systems, M.Sc. Thesis, University of Alberta, Edmonton, 1971.

2. Heaps, D.M., Easton, L.S., MacAllister, E.R., and Pallister, L.R., "Automation of a UDC Based Library for Searching Purposes", Proceedings of Second Seminar on UDC and Mechanized Information Systems, Frankfurt, 1-5 June 1970, pp 88-120, Danish Center for Documentation, Copenhagen, 1971.

3. Freeman, R.R. and Atherton, P., "File Organization and Search Strategy Using UDC in Mechanized Reference Retrieval Systems", Proceedings of the F.I.D./I.F.I.P. Joint Conference, Rome, 14-17 June 1967, pp 122-152, North-Holland Publishing Company, Amsterdam, 1968.

4. Veaner, A.B., Seminar on the SPIRES BALLOTS SYSTEM at University of Alberta, Edmonton, February 1972.

5. Mercier, M.A., Study of UDC and Other Indexing Languages through Computer Manipulation of Machine Readable Data Bases, M.Sc. Thesis, University of Alberta, Edmonton, 1972.

6. Alber, F.M., On-line Thesaurus Design for an Integrated Information System, M.Sc. Thesis, University of Alberta, Edmonton, 1972.

7. Harper, S.F., "The Universal Decimal Classification", American Documentation, V. 5, pp 195-213, 1951.

8. Mills, J., The Universal Decimal Classification, Rutgers Series of Systems for the Intellectual Organization of Information, V. 1, Graduate School of Library Service, The State University, Rutgers New Brunswick, New Jersey, 1964.

9. British Standards Institution, Guide to the Universal Decimal Classification (UDC), British Standards House, London, 1963.

10. Chambers, J., "Staking a Claim in the Arctic", The New Trail, (The University Of Alberta Alumni Magazine), pp 17-20, April 1969.

11. Roberts, B., The Organization of Polar Information, Scott Polar Research Institute Occasional Paper No. 1, 1960.

12. Roberts, B., Universal Decimal Classification for Use in Polar Libraries, Second Edition, Scott Polar Research Institute Occasional Paper No. 2, 1963.

13. Heaps, H.S., and Thiel, L.H., "Program Design for Retrospective Searches on Large Data Bases", Information Storage and Retrieval, V. 8, pp 1-20, 1972.

14. Bourne, C.P., and Ford, D.F., "A Study of Methods for Systematically Abbreviating English Words and Names", Journal of the Association for Computing Machinery, V. 8, pp 538-552, 1961.

15. Ruecking, F.H., "Bibliographic Retrieval from Bibliographic Input; the Hypothesis and Construction of a Test", Journal of Library Automation, V. 1, pp 227-238, 1968.

16. Kilgour, F.G., "Retrieval of Single Entries from a Computerized Library Catalog File", Proceedings of the American Society for Information Science, Columbus, Ohio, 20-24 Oct. 1968, V. 5, pp 133-136, Greenwood Publishing Corporation, New York, 1968.

17. Marron, B.A., and DeMaine, P.A.D., "Automatic Data Compression", Communications of the Association for Computing Machinery, V. 10, No. 11, pp 711-715, 1967.

18. Schwartz, E.S., "A Dictionary for Minimum Redundency Encoding", Journal of the Association for Computing Machinery, V. 10, pp 413-439, 1963.

19. Libertz, B.A., Stangl, P., and Taylor, K.F., "Performance of Ruecking's Word-Compression Method when Applied to Machine Retrieval from a Library Catalogue", Journal of Library Automation, V. 2, pp 266-271, 1969.

20. Nugent, W.R., "Compression Coding Techniques for Information Retrieval", Journal of Library Automation, V. 1, pp 250-260, 1968.

21. Dolby, J.L., "An Algorithm for Variable-Length Proper Name Compression", Journal of Library Automation, V. 3, pp 257-275, 1970.

22. Treleaven, R.L., Abbreviation of English Words to Standard Length for Computer Processing, M.Sc. Thesis, University of Alberta, Edmonton, 1970.

23. Huffman, D.A., "A Method for the Construction of Minimun Length Codes", Proceedings of I.R.E. , V. 40, pp 1098-1101, 1952.

24. Colombo, D.S. and Rush, J.E., "Use of Word Fragments in Computer-Based Retrieval Systems", Journal of Chemical Documentation, V. 9, pp 47-59, 1969.

25. Kucera, H. and Francis, W.N., Computational Analysis of Present-Day American English, Brown University Press, Providence, 1967.

26. Davidson, L.D., "Theory of Adaptive Data Compression", Advances in Communication, A.V. Balakrishinan, Editor, Academic Press, New York, 1966.

27. Reid, W.D., and Heaps, H.S., "Compression of Data for Library Automation", Canadian Association of College And University Libraries: Automation in Libraries 1971, (Ottawa: Canada Library Association, 1971), pp 211-221, 1971

28. Knuth, D.E., The Art of Computer Programming, V. 1, Addison-Wesley, Reading, Massachusetts, 1969.

29. Salton, G., "Manipulation of Trees in Information Retrieval", Communications of the Association for Computing Machinery, V. 5, pp 103-114, 1962.

30. Gauthier, R.L., and Ponto, S.D., Designing Systems Programs, Prentice-Hall, Englewood cliffs, 1970.

31. Dimsdale, J.J., and Heaps, H.S., "File Structrue For An On-Line Catalog Of One Million Titles", Journal of Library Automation, V. 6, pp 37-55, 1973.

32. Alber, F.M. and Heaps, D.M., "Classifying, Indexing and Searching Resource Management Information Via an On-Line Thesaurus", Proceedings of the Third Annual Meeting of the Western Canada Chapter ASIS, pp 105-116, 1971.

33. Matheson, E.,Benbow, J.A, and Natrass, B.J., "The Recovery System for the City of Edmonton's Service Order System", (In House Publication, City of Edmonton), March 1974.

34. Dobay, S.R., and Heaps, D.M., Machine-Readable Subject Authority Lists and Thesauri as Aids in Classification and Concept Analysis", Proceedings of the Fourth Annual Meeting of the Western Canada Chapter ASIS, pp 159-169, 1972.

35. Schuegraf, E.J., and Heaps, H.S., "Selection of Equifrequent Word Fragments For Information Retrieval", Information Storage and Retrieval, V. 9, pp 697-711, 1973.

APPENDIX

BOREAL INSTITUTE FOR NORTHERN STUDIES LIBRARY

INFORMATION RETRIEVAL

USER'S GUIDE

1.0 Introduction

The purpose of this information retrieval system is to provide a complete on-line automated library system for the Library of the Boreal Institute of Northern Studies.

2.0 The Command Language

The command language is a phrase structure language, defined by a formal metalanguage. This metalanguage makes use of the notation and definitions explained below.

i) A syntactic entity is denoted by its name enclosed in the brackets < and >.

ii) A syntactic rule has the form

<A> ::= X

where <A> is a member of the set of syntactic entities and X is any possible sequence of basic constituents and syntactic entities.

iii) The form <A> ::= x | y | ... | z is used as an abbreviation for the set of syntactic rules

<A> ::= x
<A> ::= y
.
.
.
<A> ::= z

iv) All words inclosed in quotes, called reserved words, are represented by the same words in capital letters without the quotes in an actual request. Reserved words may be abbreviated by typing in the underlined

portion only.

The basic constituents are defined by the following syntactic rules.

<character> ::= ␣ | ¢ | . | < | & | ! | $ |) | ; | - | /
| , | % | _ | > | ? | : | (| # | @ | ' | = | " | A
| B | C | D | E | F | G | H | I | J | K | L | M | N
| O | P | Q | R | S | T | U | V | W | X | Y | Z | 0
| Z | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<blank> ::= ␣ | ␣ <blank> (one or more blanks)

<and> ::= +

<or> ::= |

<not> ::= ¬

<logic> ::=<and> | <or> | <not>

<command word> ::= '*DELETE' | '*POINT' | '*RECORD' |
'*SEARCH' | '*SET' | '*SIGNOFF' | '*SIGNON' |
'*STATISTICS' | '*THESURUS' | '*TRANS'

<type word> ::= '*ACCESSION' | '*AUTHOR' | '*PUBLISHER' |
'*SUBJECT' | '*TITLE' | '*UDC'

<control word> ::= '*BROADER TERMS' | '*FULL' |
'*NARROWER TERMS' | '*RELATED TERMS' | '*SYNONYMS'

<output word> ::= '*FULL' | '*SMALL'

The syntactic entities are defined by the following syntactic rules.

<word> ::= <letter> | <letter> <word>

<words> ::= <word> | <word> <words>

<d-sign> ::= $ | $ <d-sign>

<q-word> ::= <word> | <word># | <word> <d-sign>

<question> ::= <q-word> | (<question>) | <q-word> <logic>
<question> | <q-word> <logic> <question> <logic>
<type word> <question> | <q-word> <logic>
(<question> <type word> <question>)

<password> ::=<letter> <letter> <letter> <letter>

<account number> ::= <letter> <letter> <letter> <letter>

The command language was designed to demonstrate the advantageous of the data structure designed during this study. The input of commands is in free-form text which may be entered through either a terminal or a card reader. The retrieval system always types out a colon, ':', when it is ready to receive input.

Precautionary steps to guard against possible destruction of information are vital for a system to which many users have access. In this system the <password> feature eliminates the possibility of an unauthorized user being allowed to alter information in system files. Further safe guarding could be included in the operating system by allowing only input from selective terminals to alter the the data base.

Note that the first two input lines of all commands that require a <password> have a restricted format. The first line must only contain the <command word> while the second line must only contain the <password>. Continuing in

line three, the remainder of the command is format free.

3.0 Commands

3.1.0 *DELETE

This command deletes the record with the given accession number from the compressed catalogue data base. All references in the inverted index to this record are also deleted.

3.1.1 Syntax of the *DELETE Command

*DELETE
<password>
<accession number> *END

3.1.2 Example of the *DELETE Command

: *DELETE
PASSWORD?
: ■■■■
: 000256 *END

3.2.0 *POINT

This command is used to alter the UDC-Subject dictionary.

3.2.1 Syntax of the *POINT command

i) Format 1

*POINT
<password>
*SUB <word> *TO *UDC <word> *END

ii) Format 2

*POINT
<password>
*UDC <word> *TO *SUB <word> *END

iii) Format 3

*POINT
<password>
*SUB <word> *FROM *UDC <word> *TO *UDC <word> *END

iv) Format 4

*POINT
<password>
*UDC <word> *FROM *SUB <word> *TO *SUB <word> *END

Format 1 alters the dictionary such that the given subject word will refer to the given UDC number. Likewise, format 2 alters the dictionary such that the given UDC number will refer to the given subject word.

Format 3 exchanges the subject word's reference from the first UDC number to the second UDC number. Similarly, format 4 exchanges the UDC number's reference from the first subject word to the second subject word.

3.2.2 Examples of the *POINT Command

```
: *POINT
PASSWORD?
: ****
: *UDC 591.54 *TO *SUB BEARS DENNING HABITS *END
```

The above example results in the UDC number 591.54 refering to the subject word 'BEARS DENNING HABITS'.

```
: *POINT
PASSWORD?
: ****
: *UDC 330.02 *FROM *SUB
: DEVELOPMENT *TO *SUB ECONOMIC
: :DEVELOPMENT
: *END
```

This command alters the UDC number '330.02' reference from the subject word 'DEVELOPMENT' to the subject word 'ECONOMIC DEVELOPMENT'

3.3.0 *RECORD

This command instructs the system to add the indicated record to the compressed data base.

3.3.1 Syntax of the *RECORD Command

i) Format 1 for non periodical holdings

*RECORD
<password>
ACC <words> (accession number)
CAL <words> (call number)
AUT <words> (author)
TIT <words> (title)
SUB <words> (subtitle)
PUB <words> (publisher)
ABS <words> (abstract)
YEA <words> (year)
PAG <words> (pages)
FOR <words> (book format)
SER <words> (series)
BIB <words> (bibliography)
GLO <words> (glossary)
LCN <words> (LC number)
ISB <words> (ISB number)
GDC <words> (GDC number)

```
ORD <words>      (order number)
UDC <words>      (UDC number)
ANA <words>      (anayltical authors)
*END
```

ii) Format 2 for periodicals

```
*RECORD
<password>
ACC <words>      (accession number)
PTI <words>      (title)
PPU <words>      (publisher)
PDA <words>      (date of first issue)
PAD <words>      (additional information)
PLA <words>      (issues in library)
PVO <words>      (volume number)
PIS <words>      (issue number)
PYE <words>      (date of issue)
PAN <words>      (analytic author)
*END
```

The descriptors in parenthesis are not part of the syntax but are included to indicate the contents of each entry. Note that inclusion of any one of the entries is optional. Also optional is the order the entries appear in the command.

All entries must start on a new line with its keyword followed by a blank in the first four columns of the line. Continuation lines are indicated by four blanks in the first four columns of the line. The underscore character '_' must be entered instead of a blank in a word.

3.3.2 Example of the *RECORD Command

```
: *REC
PASSWORD?
: ■■■■
: ACC 05678
: CAL 91(091):(7)_CUM
: AUT CUMMING,W.P., SKELTON,R.A.  AND QUINN,D.D.
: TIT THE DISCOVERY OF NORTH AMERICA
: PUB MCCLELLAND AND STEWART
: ABS DISCOVERY AND EXPLORATION OF THE
:     NORTH AMERICAN CONTINENT, FROM
:     EARLIEST REFERENCES TO THE FIRST
:     PERMANENT SETTLEMENTS AS DESCRIBED
:     BY THE EXPLORERS AND DELINEATED
:     BY CONTEMPORARY EUROPEANS.
: YEA C1971
: PAG 304P
: FOR 31CM ILLUS,MAPS
: BIB 298-300
: *END
```

The above command will add to the data base the indicated holding. The inverted indices for all searchable words will be updated to faciliate searching for this new record. Likewise, for all new words that do not appear in the dictionary, a request is made to confirm the addition of the new word into the dictionary. This conformation is made to protect against filling the dictionary with incorrectly spelled words.

3.4.0 *SEARCH

The *SEARCH command specifies a search of holdings in the compressed data base. The search can be made on accession numbers, authors, publishers, subject words, title words, UDC numbers, or any combination of the above types.

The logic symbols +, |, and ¬ represent the logic function of conjunction (AND), disjunction (OR), and negation (NOT) respectively.

The two types of right truncation which may be specified by a search word are limited and unlimited right truncation. An unlimited right truncated search word ends in a #. This indicates to the search program to retrieve all records referenced by words with a left root of the given search word with zero or more additional letters on the right. A limited right truncated search word ends with one or more $. This requests a retrieval of all records

referenced by words with a left root of the given search word with from zero to n additional letters on the right,. Where n is the number of $'s that appear on the right of the search word. For example the search word government$ would retrieve all records referenced by either government or governments; while the search word government# would retrieve all records referenced by any one of the words government, governments, governmental, or governmentally.

3.4.1 Syntax of the *SEARCH Command

i) Format 1

*SEARCH <type word> <question> *END

ii) Format 2

*SEARCH *FULL <type word> <question> *END

iii) Format 3

*SEARCH <type word> <question> *STATISTICS *END

iv) Format 4

*SEARCH *FULL <type word> <question> *STATISTICS *END

The reserved word '*FULL' controls the format of the output of the records in the compressed data base which satisfy the search question. The use of the word *FULL indicates that the user wishes the complete record to be written out. The omission of *FULL would indicate that the user only requests an abbreviated form of the record to be

written out. This smaller output would include the accession number, call number, author, title, abstract, and year of publication. This abbreviated form allows the average user to receive the pertinent information about a holding at a cheaper and faster rate then by requesting a full output.

The *STATISTICS command word given after a search question requests analytic statistics of the search question. The statistics returned include the number of records retrieved by each search word and the number of records retrieved by each level of the search question.

3.4.2 Examples of the *SEARCH Command

```
: *SEARCH *FULL *TITLE ANTARCTIC @ ( *AUT
: WALLEN,I.E | LLANO,G.A ) *END
```

This example requests a full listing without statistics of all holdings with a title word of 'Antarctic' and author of 'Wallen, I.E' or 'Llano,G.A'

```
: *SEARCH *PUB CLARKE,IRWIN & CO
: | AMERICAN GEOPHYSICAL UNION
: | ( MCGRAW-HILL @ *ACC 000579 ) *STAT *END
```

This command requests the short listing with statistics of all holdings that have as publishers 'Clarke,

Irwin & Co' or 'American Geophysical Union' or has publisher 'McGraw-Hill' and accession number '000579'.

3.5.0 *SET

The *SET command changes the password.

3.5.1 Syntax of the *SET Command

*SET <password> <password> *END

3.5.2 Examples of the *SET Command

```
: *SET
PASSWORD?
: ■■■■
: NEW *END
```

The result of this example would change the password to 'NEW'.

3.6.0 *SIGNOFF

The *SIGNOFF command indicates to the system that the current user is finished processing commands.

3.6.1 Syntax of the *SIGNOFF Command

*SIGNOFF

3.7.0 *SIGNON

The *SIGNON command indicates that a user wishes to start to use the system.

3.7.1 Syntax of the *SIGNON Command

*SIGNON <account number>

The account number can be used to control the use of the system or the support an accounting system which charges for the use of the system.

3.8.0 *THESAURUS

This command allows the user to obtain a thesaurus from the UDC schedules.

3.8.1 Syntax of the *THESAURUS Command

*THESAURUS <control word> <word> *END

The <control word> indicates the type of output desired by the user.

3.8.2 Examples of the *THESAURUS Command

: *THESAURUS *FULL economic development
: natural resources *END

: *THE *NAR economic development natural resources
*END

The first example requests a full thesaurus for the word 'economic development natural resources'. While the second example requests only the narrower terms.

3.9.0 *TRANS

The *TRANS command translates a UDC number or subject word to the subject words or UDC number which they respectively refer to.

3.9.1 Syntax of the *TRANS Command

i) Format 1

*TRANS *SUBJECT <word> *END

ii) Format 2

*TRANS *UDC <word> *END

3.9.2 Examples of the *TRANS Command

: *TRANS *UDC 330.02 *END

: *TRANS *SUB BEARS DENNING HABITS *END

The first transaction lists all subject words which correspond to the UDC number 330.02. The second transaction lists all UDC numbers which correspond to the subject word 'bears denning habits'.

4.0 Prompting

There is an extensive prompting feature built into the retrieval system. If the librarian or the patron is entering a command and is unsure of what is required next, then he can type in a question mark, ?. The system will then type out a message to indicate what data should be entered next.

5.0 Errors

If a syntactically incorrect command is entered, the program will indicate the error. After an error, the user must re-enter the complete command.

The following error messages may appear.

IMPROPER TYPE

Invalid <type word>, <control word>, or <output word>.

IMPROPER COMMAND

Invalid <command word>.

IMPROPER NUMBER OF PARAMETERS

The *POINT command entered is syntactically incorrect.

IMPROPER RECORD FIELD

a field descriptor in the *ADD command is incorrect.

<WORD>
IS TO BE ADDED TO DATABASE
PLEASE CONFIRM "OK" OR "CANCEL"

The indicated word does not appear in the appropriate dictionary. A response of OK will cause the addition of this word to the dictionary a response of CANCEL will result in the word not being added.

NO TRANSLATION IN DICTIONARY

The requested word indicated in a search has no corresponding UDC number.

PASSWORD INCORRECT, TRY AGAIN

Invalid <password> entered, re-enter the password.

INVALID PASSWORD

This was the third consecutive time an invalid <password> was entered. Re-enter the command.